

AD-A191 327

GENERALIZING THE STRUCTURE OF EXPLANATIONS IN
EXPLANATION-BASED LEARNING(U) ILLINOIS UNIV AT URBANA
COORDINATED SCIENCE LAB J M SHAYLIK DEC 87

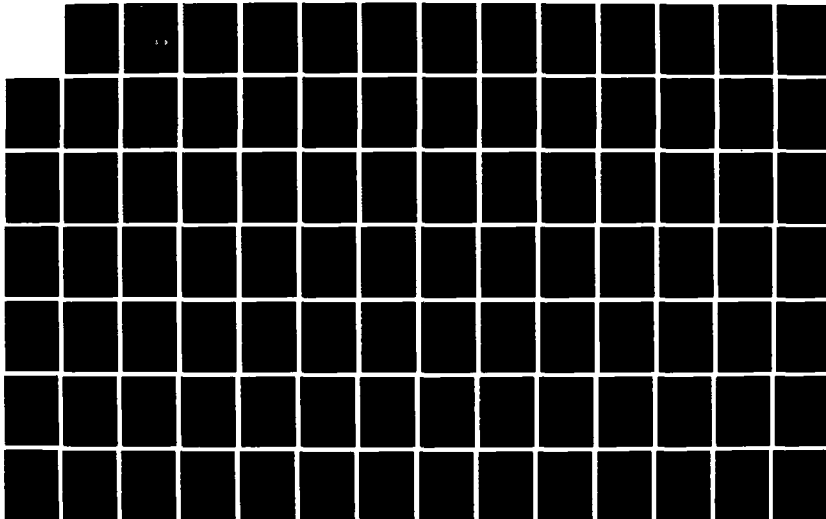
1/4

UNCLASSIFIED

UILU-ENG-87-2276 N00014-86-K-0309

F/G 23/2

NL





1.0



1.1



1.25



2.8



3.15



3.5



4.0



4.5



5.0



5.6



6.3



7.1



8.0



9.0



10.0



1.4



1.6

December 1987

DTIC FILE COPY

UILU-ENG-87-2276

2

COORDINATED SCIENCE LABORATORY
College of Engineering

AD-A191 327

GENERALIZING THE STRUCTURE OF EXPLANATIONS IN EXPLANATION-BASED LEARNING

DTIC
ELECTE
FEB 23 1988
S D
CD

Jude William Shavlik

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Approved for Public Release. Distribution Unlimited.

88 2 23 011

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILLU-ENG-87-2276		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (if applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research National Science Foundation	
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801		7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy St., Arlington, VA 22217 1800 G. Street N.W., Washington, D.C. 20550	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION ONR/NSF	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0309 NSF IST 85-11542	
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy St., Arlington, VA 22217 1800 G. Street N.W., Washington, D.C. 20550		10. SOURCE OF FUNDING NUMBERS	
		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Generalizing the Structure of Explanations in Explanation-Based Learning			
12. PERSONAL AUTHOR(S) Shavlik, Jude William			
13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day) December 1987	15. PAGE COUNT 289
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) <p>Explanation-based learning is a recently developed approach to concept acquisition by computer. In this type of machine learning, a specific problem's solution is generalized into a form that can later be used to solve conceptually similar problems. A number of explanation-based generalization algorithms have been developed. Most do not alter the structure of the explanation of the specific problem — no additional objects nor inference rules are incorporated. Instead, these algorithms generalize by converting constants in the observed example to variables with constraints important concepts, in order to be properly learned, require that the structure of explanations be generalized. This can involve generalizing such things as the number of entities involved in a concept or the number of times some action is performed. For example, concepts such as momentum and energy conservation apply to arbitrary numbers of physical objects, clearing the top of a desk can require an arbitrary number of object relocations, and setting a table can involve an arbitrary number of guests.</p> <p>Two theories of extending explanations during the generalization process have been developed, and computer implementations have been created to computationally test these approaches. The Physics101 system utilizes characteristics of mathematically-based problem solving to extend mathematical calculations in a psychologically-plausible way, while the BAGGER system implements a domain-independent approach to generalizing explanation structures. Both of these systems are described and the details of their algorithms presented. Several examples of learning in each system are discussed. An over,</p>			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL		22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

19 approach to the *operationality/generalizability* trade-off and an empirical analysis of explanation-based learning are also presented. The computer experiments demonstrate the value of generalizing explanation structures in particular, and of explanation-based learning in general. These experiments also demonstrate the advantages of learning by observing the intelligent behavior of external agents. Several open research issues in generalizing the structure of explanations and related approaches to this problem are discussed. This research brings explanation-based learning closer to its goal of being able to acquire the full concept inherent in the solution to a specific problem.

© Copyright by
Jude William Shavlik
1988



Account For	
DTIC ORA&I	<input checked="checked" type="checkbox"/>
DTIC T&B	<input type="checkbox"/>
DTIC I&I	<input type="checkbox"/>
DTIC J&J	<input type="checkbox"/>
DTIC K&K	<input type="checkbox"/>
DTIC L&L	<input type="checkbox"/>
DTIC M&M	<input type="checkbox"/>
DTIC N&N	<input type="checkbox"/>
DTIC O&O	<input type="checkbox"/>
DTIC P&P	<input type="checkbox"/>
DTIC Q&Q	<input type="checkbox"/>
DTIC R&R	<input type="checkbox"/>
DTIC S&S	<input type="checkbox"/>
DTIC T&T	<input type="checkbox"/>
DTIC U&U	<input type="checkbox"/>
DTIC V&V	<input type="checkbox"/>
DTIC W&W	<input type="checkbox"/>
DTIC X&X	<input type="checkbox"/>
DTIC Y&Y	<input type="checkbox"/>
DTIC Z&Z	<input type="checkbox"/>
A-1	

**GENERALIZING THE STRUCTURE OF EXPLANATIONS
IN EXPLANATION-BASED LEARNING**

BY

JUDE WILLIAM SHAVLIK

B.S., Massachusetts Institute of Technology, 1979

B.S., Massachusetts Institute of Technology, 1979

M.S., Yale University, 1980

THESIS

**Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988**

Urbana, Illinois

GENERALIZING THE STRUCTURE OF EXPLANATIONS IN EXPLANATION-BASED LEARNING

Jude William Shavlik, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1988
Gerald Francis DeJong, Advisor

Explanation-based learning is a recently developed approach to concept acquisition by computer. In this type of machine learning, a specific problem's solution is generalized into a form that can later be used to solve conceptually similar problems. A number of explanation-based generalization algorithms have been developed. Most do not alter the structure of the explanation of the specific problem — no additional objects nor inference rules are incorporated. Instead, these algorithms generalize by converting constants in the observed example to variables with constraints. However, many important concepts, in order to be properly learned, require that the *structure* of explanations be generalized. This can involve generalizing such things as the number of entities involved in a concept or the number of times some action is performed. For example, concepts such as momentum and energy conservation apply to arbitrary numbers of physical objects, clearing the top of a desk can require an arbitrary number of object relocations, and setting a table can involve an arbitrary number of guests.

Two theories of extending explanations during the generalization process have been developed, and computer implementations have been created to computationally test these approaches. The **Physics 101** system utilizes characteristics of mathematically-based problem solving to extend mathematical calculations in a psychologically-plausible way, while the **BAGGER** system implements a domain-independent approach to generalizing explanation structures. Both of these systems are described and the details of their algorithms presented. Several examples of learning in each system are discussed. An approach to the *operationality/generality* trade-off and an empirical analysis of explanation-based learning are also presented. The computer experiments demonstrate the value of generalizing explanation structures in particular, and of explanation-based learning in general. These experiments also demonstrate the advantages of learning by observing the intelligent behavior of external agents. Several open research issues in generalizing the structure of explanations and related approaches to this problem are discussed. This research brings explanation-based learning closer to its goal of being able to acquire the full concept inherent in the solution to a specific problem.

Dedication

*To my parents — Elizabeth and Raymond — for their love and support
during more than two decades of education.*

Acknowledgements

The nearly four years I have spent in the artificial intelligence group of the Coordinated Science Laboratory have been both intellectually rewarding and personally enjoyable. Foremost credit is due Professor Gerald DeJong, whose gently guiding leadership, along with that of Professors David Waltz and Robert Stepp, has created a stimulating intellectual environment. Jerry DeJong has been an ideal advisor. I am grateful for both his providing me the freedom to pursue my interests and for his insightful comments that lead me to follow fruitful research paths. Many of the best ideas in this thesis resulted from thought-provoking discussions with him. He has been an excellent role model, teaching me what it means to be a creative researcher.

Over these last four years I have learned much from discussions with the other members of the explanation-based learning group. Interactions with Scott Bennett, Steven Chien, Melinda Gervasio, Raymond Mooney, Paul O'Rorke, Shankar Rajamoney, Ashwin Ram, and Alberto Segre have stimulated many interesting ideas. Special credit is due Ray Mooney. Discussions with him have been quite enlightening in the past, and I anticipate they will continue to be so in the future.

Along with Professor DeJong, Professors Gregg Collins, Kenneth Forbus, Dedre Gentner, Brian Ross, and Robert Stepp served on my final examination committee. Professor Nachum Dershowitz served on my preliminary examination committee. Their insights on learning, from the perspectives of both computer science and psychology, have been educational. In addition to being thankful for their comments and suggestions regarding my research, I have enjoyed my interactions with them in classes and seminars. I will miss the vibrant interdisciplinary community of cognitive science researchers they have helped create.

My officemates over the years deserve commendation. Marcy Dorfman, Paul O'Rorke, Robert Reinke, and Koichi Tanaka have provided perceptive comments on numerous topics, and it has been a pleasure sharing an office with each of them.

The discussions at our weekly machine learning seminar have produced in-depth analyses of various approaches to learning. The contributions of the additional participants are gratefully recognized: Brian Falkenhainer, Lawrence Holder, Bartlett Mel, Bharat Rao, and Bradley Whitehall.

Staff members Frances Bridges, David Ballman, and Catherine Cassells have provided invaluable support, for which I thank them.

Finally, I wish to thank Zoann Branstine for being more than a close friend. Her love and understanding are greatly appreciated.

This research was partially supported by the Office of Naval Research under grant N00014-86-K-0309, by the National Science Foundation under grant NSF IST 85-11542, and by a University of Illinois Cognitive Science/Artificial Intelligence Fellowship. A forgivable loan from the General Electric Corporation is also gratefully acknowledged.

Table of Contents

Chapter	Page
1 Introduction	1
1.1. The Need for Generalizing the Structure of Explanations	3
1.2. Explanation-Based Learning	5
1.3. The Physics 101 System	9
1.4. The BAGGER System	12
1.5. Notational Conventions	13
1.6. Thesis Organization	13
2 Overview of the Physics 101 System	15
2.1. The Learning Model	17
2.2. Some Terminology	18
2.3. Obstacles	19
2.4. Special-Case Schemata	21
2.5. Initial Knowledge of the System	22
2.6. Summary	26
3 Building Explanations in Physics 101	28
3.1. A Sample Problem	29
3.2. Verifying a Teacher's Solution	30
3.3. Explaining Solutions	34
3.4. Understanding Obstacles	37
3.5. Summary	40
4 Generalizing Solutions in Physics 101	42
4.1. Using the Cancellation Graph to Guide Generalization	42
4.2. The Result of Standard Explanation-Based Learning	49
4.3. Why the Specific and Generalized Calculations can Differ	51
4.4. Learning Special-Case Schemata	53
4.5. Performance Analysis	61
4.6. Other Approaches to Learning in Mathematical Domains	63
4.7. Summary	65
5 Physics 101's Problem Solver	68
5.1. Schema-Based Problem Solving	71
5.2. Choosing the Initial Equation	72

Chapter	Page
5.3. Transforming an Expression into an Acceptable Form	75
5.4. Summary	82
6 Algorithmic Details of Physics 101	85
6.1. Constructing the Cancellation Graph	86
6.2. Using the Cancellation Graph	94
6.3. Problem Solving in Physics 101	100
6.4. Summary	107
7 Additional Physics 101 Examples	108
7.1. Learning about Energy Conservation	108
7.2. Learning about the Sum of Internal Forces	112
7.3. Using the New Force Law to Learn about Momentum	117
7.4. Attempting to Learn from a Two-Ball Collision	118
7.5. Summary	120
8 Overview of the BAGGER System	122
8.1. Some Sample Problems	123
8.2. Situation Calculus	127
8.3. Sequential Rules	128
8.4. Representing Sequential Knowledge	129
8.5. Summary	130
9 Generalization in BAGGER	132
9.1. The BAGGER Generalization Algorithm	132
9.2. Problem Solving in BAGGER	144
9.3. Simplifying the Antecedents in Sequential Rules	145
9.4. Summary	150
10 Details of Several BAGGER Examples	153
10.1. Building a Tower	154
10.2. More Tower Building Rules	162
10.3. Clearing an Object	167
10.4. Setting a Table	177
10.5. A General Version of DeMorgan's Law	181
10.6. Summary	183

Chapter	Page
11 Empirical Analysis of Explanation-Based Learning Algorithms	184
11.1. Experimental Methodology	185
11.2. Comparison of the Two Training Strategies	189
11.3. Effect of Increased Problem Complexity	197
11.4. Time Spent Learning	201
11.5. Operationality/Generality Trade-Off	206
11.6. Rule Access Strategies	211
11.7. Combining sEBL and BAGGER	221
11.8. Clearing Blocks	225
11.9. Additional Experimental Results	229
11.10. Summary	237
12 Conclusion	240
12.1. Relation to Other Work	242
12.2. Some Open Research Issues	245
12.3. Final Summary	253
Appendix A BAGGER's Initial Inference Rules	256
Appendix B Standard Deviations of Experimental Results	263
References	266
Vita	280

Chapter 1

Introduction

The ability to learn is fundamental to intelligent behavior. Extracting the general concept inherent in the solution to a specific problem is a powerful way to learn. Often the solution to a specific problem will repeatedly employ a technique or collection of techniques. It is an important, but difficult, problem in machine learning to correctly generalize such a sequence once observed. Sometimes the number of repetitions itself should be the subject of generalization. Other times it is quite inappropriate to alter the number of repetitions. A recently developed paradigm of machine learning, called *explanation-based learning* (EBL) [DeJong86, Ellman87, Mitchell86], provides an approach to this issue. In this type of learning, a specific problem solution is generalized into a form that can be later used to solve conceptually similar problems. The generalization process is driven by the *explanation* of why the solution worked. Knowledge about the domain allows the explanation to be developed, and then generalized. The explanation of a technique's functionality dictates when it is valid and proper to generalize the number of times it occurs.

This thesis addresses the important issue in explanation-based learning of *generalizing the structure of explanations*. This can involve generalizing such things as the number of entities

involved in a concept or the number of times some action is performed. Generalizing the structure of the explanation has been largely ignored in previous explanation-based learning research. Instead, other research has focused on changing constants into variables and determining the general constraints on those variables.

The central claim of this thesis is:

Explanation structures that suffice for understanding a specific example are not always satisfactory for generalizing the example. The explanation structure must often be augmented if a useful generalization is to be produced.

Evidence for this claim and techniques for augmenting explanation structures are presented.

While generalizing the structure of an explanation can involve more than generalizing the number of times a technique is employed — for example, the *order* techniques are applied or the actual techniques used can be generalized, this thesis largely focusses on the topic of *generalizing number*. Usually this involves generalizing a fixed number of applications of a technique into a possibly constrained but unbounded number of applications. The phrase *generalizing to N* is also used to indicate this process.

Generalizing number, like more traditional generalization in explanation-based learning, results in the acquisition of a new inference rule. The difference is that the sort of rule that results from generalizing number describes the situation after an indefinite number of world changes or other inferences have been made. Each such rule subsumes a potentially infinite class of rules acquired by standard explanation-based generalization techniques. Thus, with such rules the storage efficiency can be dramatically improved, the expressive power of the system is increased, and, as shown in chapter 11, the system's performance efficiency can also be higher than without these rules.

The next section in this chapter further motivates the need to generalize explanation structures. Following that, a lengthier introduction to explanation-based learning is provided. The subsequent sections introduce two implemented systems that address the need to generalize explanation structures. The remaining chapters of this thesis further elaborate these two systems. This chapter closes with comments on the notation used in the rest of the thesis and a brief description of the remaining chapters.

1.1. The Need for Generalizing the Structure of Explanations

The need for generalizing the structure of explanations can be seen by considering two existing explanation-based learning systems. The LEAP system [Mitchell85] is shown an example of using *NOR* gates to compute the boolean *AND* of two *OR*'s. It discovers that the technique generalizes to computing the boolean *AND* of any two inverted boolean functions. However, LEAP cannot generalize this technique to allow constructing the *AND* of an arbitrary number of inverted boolean functions using a multi-input *NOR* gate. This is the case even if LEAP's initial background knowledge were to include the general version of DeMorgan's Law and the concept of multi-input *NOR* gates. Generalizing the number of functions requires alteration of the original example's explanation.

Ellman's system [Ellman85] also illustrates the need for generalizing number. From an example of a four-bit circular shift register, his system constructs a generalized design for an arbitrary four-bit permutation register. A design for an N -bit circular shift register cannot be produced. As Ellman points out, such generalization, though desirable, cannot be done using the technique of changing constants to variables.

Many important concepts, in order to be properly learned, require generalization of number. For example, physical laws such as momentum and energy conservation apply to arbitrary numbers of objects, constructing towers of blocks requires an arbitrary number of repeated stacking actions, and setting a table involves a range of possible numbers of guests.¹ In addition, there is recent psychological evidence [Ahn87] that people can generalize number in an explanation-based fashion.

Repetition of an action is not a sufficient condition for generalization to N to be appropriate. Compare two simple examples. Generalizing to N is necessary in one but inappropriate in the other. The examples are:

- observing a previously unknown method of moving an obstructed block, and
- seeing, for the first time, a toy wagon being built.

¹ These three concepts are among those acquired by the systems described in this thesis. Details of their acquisition are provided in the following chapters.

The initial states of the two problems appear in figure 1.1. Suppose a learning system observes an expert achieving the desired states. In each case, consider what general concept should be acquired.

In the first example, the expert wishes to move, using a robot manipulator, a block which has four other blocks stacked in a tower on top of it. The manipulator can pick up only one block at a time. The expert's solution is to move all four of the blocks in turn to some other location. After the underlying block has been cleared, it is moved. In the second example, the expert wishes to construct a movable rectangular platform, one that is stable while supporting any load whose center of mass is over the platform. Given the platform and a bin containing two axles and four wheels, the expert's solution is to first attach each of the axles to the platform. Next all four of the wheels are grabbed in turn and mounted on an axle protrusion.

This comparison illustrates an important problem in explanation-based learning. Generalizing the block unstacking example should produce a plan for unstacking *any* number of obstructing blocks, not just four as observed. The wagon-building example, however, should not generalize the number "4." It makes no difference whether the system is given a bin of five, six, or 100 wheels, because only four wheels are needed to fulfill the functional requirements of a stable wagon.

Standard explanation-based learning algorithms [DeJong86, Fikes72, Hirsh87, Kedar-Cabelli87, Mitchell86, Mooney86, O'Rourke87a] and similar algorithms for chunking [Laird86] cannot treat these cases differently. These algorithms, possibly after pruning the explanation to eliminate irrelevant parts, replace constants with constrained variables. They cannot



Figure 1.1 Initial States for Two Sample Problems

significantly augment the explanation during generalization. Thus, the *building-a-wagon* type of concept will be correctly acquired but the *unstacking-to-move* concept will be undergeneralized. The acquired schema will have generalized the identity of the blocks so that the target block need not be occluded by the same four blocks as in the example. Any four obstructing blocks can be unstacked. However, there must be exactly four blocks.² Unstacking five or more blocks is beyond the scope of the acquired concept.

Note that EBL systems do not work correctly on the *building-a-wagon* kind of problems either — they just get lucky. They do nothing to augment explanation structures during generalization. It just happens that to acquire a schema to build a wagon, *not* generalizing the explanation structure is the appropriate thing to do.

One can, of course, simply define the scope of EBL-type systems to exclude the *unstacking-to-move* concept and those like it. This is a mistake. First, the problem of augmenting the explanation during generalization, once seen, is ubiquitous. It is manifested in one form or another in most real-world domains. Second, if one simply defines the problem away, the resulting system could never guarantee that any of its concepts were as general as they should be. Even when such a system correctly constructed a concept like the *building-a-wagon* schema, it could not know that it had generalized properly. The system could not itself tell which concepts fall within its scope and which do not.

1.2. Explanation-Based Learning

In explanation-based learning, knowledge about a domain is used to construct an explanation of how a specific problem can be solved.³ The resulting explanation is then generalized so that conceptually similar problems can be quickly solved in the same manner.

The view that analyzing the solution to a single, specific problem may lead to a useful general problem-solving technique can be traced back to the STRIPS system [Fikes72]. This

² The SOAR system [Laird86] would seem to acquire a number of concepts which together are slightly more general. As well as a new operator for moving four blocks, the system would acquire new operators for moving three blocks, two blocks, and one block, but not for five or more.

³ The notion of a *problem* should be taken rather broadly. EBL can be used to justify why a specific example is a member of some concept as well as explain why a sequence of actions produces a desired state in the world.

influential system learned *macro-operators* by generalizing an operator sequence that satisfied some goal in the blocks world. Additional early work, either intentionally within the emerging paradigm of explanation-based learning or research that can now be viewed as being in the spirit of EBL, includes [DeJong81, DeJong83, Mitchell83a, Silver83, Soloway78, Sussman73, Winston83]. All of this work investigated having a computer system apply its understanding of a domain to the process of learning by generalizing solutions to specific problems.

This approach to learning can be contrasted to the more traditional approach of analyzing descriptions of a number of positive, and possibly negative, instances of some concept. In this paradigm, learning usually occurs by extracting the commonalities among the positive examples to produce a description of the concept. Care must be taken to avoid including any of the negative instances in the acquired concept. This type of learning has been termed *similarity-based learning* [Lebowitz86], although some researchers prefer *similarity/difference-based learning* because differences between the positive and negative examples can be as important as the similarities among the positive examples. Examples of research investigating this approach to machine learning include [Hunt66, Langley81, Lebowitz80, Michalski80, Mitchell78, Quinlan79, Schank82, Stepp84, Vere78, Winston75]. Reviews, comparisons, and critiques can be found in [Angluin83, Bundy85, Dietterich82, Dietterich83, Michalski83, Mitchell82, Schank86]. Connectionist approaches to learning [Rumelhart86] can also be viewed as belonging to this paradigm.

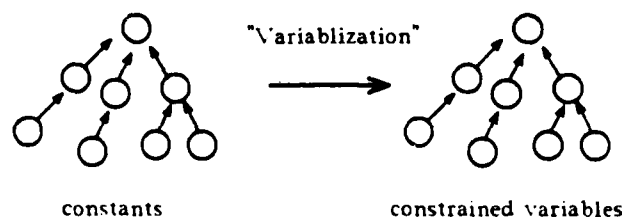
The main difference between the similarity-based and explanation-based approaches is that the similarity-based approaches do not use knowledge about a domain to justify which of the features of an example explain why it is or is not a member of the concept being learned. This can be a strength if this knowledge is not available [Dietterich86]. However, often the knowledge is available. Research in explanation-based learning is investigating how to take advantage of this additional knowledge, mitigating the need for a large number of examples and avoiding the combinatorially expensive process of searching through a large space of possible concept descriptions.

Following the initial work on explanation-based learning, a second wave of research tested and extended the emerging methodologies in more complicated domains: mathematics [Mahadevan85, O'Rourke84, Porter85, Utgoff84], circuit design [Elilman85, Mitchell85], natural language [Mooney85, Pazzani85], robotics [Segre85], physics [Shavlik85a], chemistry,

[Rajamoney85], and game playing [Minton84]. More recently, the fundamental underpinnings of explanation-based learning have become apparent and a third wave of research has focussed on developing domain-independent explanation-based generalization algorithms [DeJong86, Hirsh87, Kedar-Cabelli87, Mitchell86, Mooney86, O'Rorke87a, Rosenbloom86, Shavlik87d]. Initially the attempts to produce explanation-based generalization algorithms were influenced by the notions of *goal regression* [Waldinger77] and *weakest preconditions* [Dijkstra76]. However, more recently the simpler notion of *unification* has been seen as sufficient [Hirsh87, Kedar-Cabelli87, Mooney86].

Figure 1.2 schematically compares the operation of standard EBL algorithms and a structure-generalizing EBL algorithm. Both algorithms assume that, in the course of solving a problem, a collection of pieces of general knowledge (e.g., inference rules, rewrite rules, or plan

Standard EBL



Desired EBL

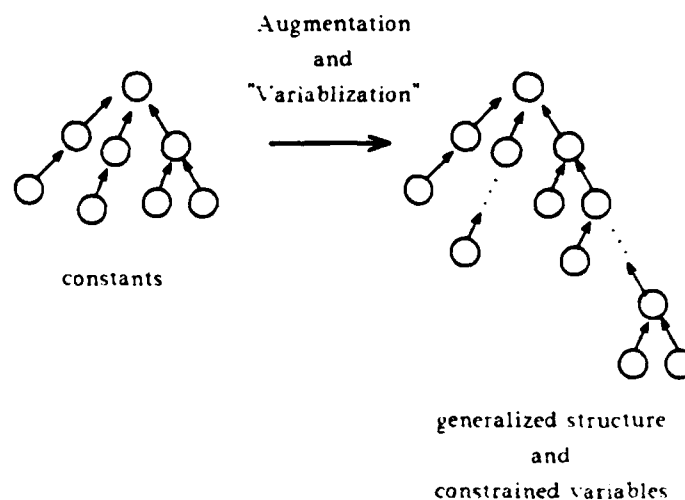


Figure 1.2 Generalizing the Structure of Explanations

schemata) are interconnected, using unification to insure compatibility. Unlike the result of standard explanation-based learning, the generalized structure in the lower right corner represents a *class* of potential explanation structures and not a single fixed structure.

The structure-generalizing algorithms presented in this thesis extend the EGGS algorithm [Mooney86], a standard domain-independent explanation-based generalization algorithm. In EGGS, the resulting explanation structure is generalized by first stripping away the details of the specific problem and then determining the most general unifier that allows the general pieces of knowledge to be connected in the same way. This involves replacing the constants in the specific explanation with constrained variables. The result is a new composite knowledge structure that contains the unifications that must hold in order for the knowledge pieces to be combined in the given way. If the leaf nodes can be satisfied, the root (goal) node will also be satisfied. There is no need to again reason about combining the pieces of knowledge together to achieve the goal. Since a substantial amount of work can be expended constructing the original solution, the new knowledge structure can lead more rapidly to a solution. The other domain-independent explanation-based generalization algorithms operate basically the same. See [Mooney86, Mooney88] for further comparison.

When generalizing an explanation structure, the necessary unifications are determined. However, importantly, the explanation structure is also reformulated so that additional pieces of knowledge are incorporated.

An important issue in explanation-based learning, one that has received little attention, is the question of how representative a sample problem is of future problems likely to be encountered. Most research assumes that each sample problem is not anomalous, and, hence, its generalization is worth saving because it will prove applicable to similar problems in the future. Often the criterion for learning is simply that the system could not solve the sample problem on its own. However care must be taken to prevent a system from being overloaded with acquired concepts that will rarely be used. This issue of deciding when to learn is exacerbated when generalizing the structure of explanations, because the final result is, roughly speaking, conceptually further away from the specific example. It is not appropriate to generalize structure whenever possible. The two systems described in this thesis conservatively estimate when generalizing the structure of explanations is appropriate. Determining the appropriateness

of generalizing structure is a theme interwoven throughout this thesis and is also seen as an important open research issue.

The next two sections introduce two approaches to the problem of generalizing the structure of explanations. In these sections, the focus of each system is described. As is typical in the explanation-based learning paradigm, both of these systems address the acquisition of structured knowledge chunks. In this thesis, these chunks of knowledge are termed *schemata* [Chafe75], and are similar in spirit to *scripts* [Cullingford78, Schank77], *frames* [Charniak76, Minsky75] and *macro-operators* [Fikes72]. New schemata are built by usefully organizing other schemata. The two approaches focus on learning composite schemata useful in problem solving. These composite schemata possess the desirable property that once one is selected, several problem-solving steps can be carried out without the need for intervening search.

1.3. The Physics 101 System

The need for generalizing number in explanation-based learning was first articulated in [Shavlik85b], an early report on the **Physics 101** system. This system learns such concepts as the general law of conservation of momentum (which is applicable to an arbitrary collection of objects) by observing and analyzing the solution to a specific three-body collision. **Physics 101** reasons about using mathematical formulae to solve problems in mathematically-based domains, learning new concepts about these domains in the process.

Mathematically-based domains are an area where the strengths of explanation-based learning are particularly appropriate, because explanation-based learning supports the construction of complicated concepts by analyzing how smaller concepts can be pieced together to solve a specific problem. This process of combining small concepts to form larger ones is the basis of progress in mathematical domains.

Understanding and generalization in **Physics 101** are guided by analyzing the elimination of *obstacles* in a specific problem. Obstacles are variables that preclude the direct evaluation of the unknown. Cancelling these variables allows the determination of the value of the unknown. One important feature of analyzing variable cancellation is that the generalization of number is properly motivated. Generalization occurs by performing the general versions of these cancellations in the general world.

The new calculation produced during generalization is often substantially more general, in terms of its structure as well as its variables, than the specific calculation. The number of entities, the identity of the operations performed, and the formulae applied may all be generalized. **Physics 101** offers a different perspective on the process of explanation-based generalization. Rather than directly using the explanation of the specific problem's solution, the explanation of a specific calculation closely guides the construction of a general version of the calculation, from which a new general concept may be extracted. No problem-solving search is performed during the construction of the general calculation — construction deterministically follows from the specific calculation. Hence, the process is relatively efficient.

The results obtained in the development of **Physics 101** are applicable to several research areas in addition to that of generalizing the structure of explanations. The next several paragraphs view **Physics 101** from the perspective of these additional topics within artificial intelligence.

Learning Apprentices

Symbolic mathematics systems, such as **MACSYMA** [Mathlab83], **MAPLE** [Geddes82], **REDUCE** [Hearn84], and **SMP** [Wolfram83], perform remarkable feats. Unfortunately these systems do not help in formulating an approach to a problem, improve their performance with experience, automatically adapt to the idiosyncrasies of individual users, nor provide comprehensible explanations of their problem-solving steps. Largely this is because the bulk of their mathematical knowledge is implicitly encoded within their algorithms (see [Fateman85] for a discussion of this).

Physics 101 is given descriptions of problems and, whenever possible, produces a solution, explaining how and why its solution works. When the system cannot solve a problem, it asks to observe a solution performed by its human user. The solution produced is analyzed and, if proved correct, generalized. The generalized problem-solving technique is then added to the system's knowledge base, thereby improving its future problem-solving performance. A learning system such as **Physics 101** can be incorporated into systems that perform symbolic mathematical computations to construct an expert aid for workers in mathematically-based domains [Shavlik86a]. By initially observing the use of composite concepts — those derivable from a domain's first principles — to efficiently solve problems, these concepts can be acquired

by the system and used to rapidly solve similar problems in the future. In this vein, **Physics 101** can be viewed as a *learning apprentice* for domains based on mathematical calculation.

Learning apprentices have been defined [Mitchell85] as

interactive knowledge-based consultants that directly assimilate new knowledge by observing and analyzing the problem-solving steps contributed by their users in their normal use of the system.

The incorporation of learning apprentices in practical problem solvers [Hill87, Minton86, Mitchell85, O'Rourke87a, Segre87a, Shavlik87b, Wilkins86] is an active area of research in machine learning. Chapter 11 presents empirical results relevant to the design of learning apprentices.

Psychologically-Plausible Modelling

Physics 101 is intended to be a psychologically-plausible model of problem solving and learning in mathematically-based domains [Shavlik85b, Shavlik86b, Shavlik87e]. It is proposed that focussing on obstacle cancellation provides a way of understanding much of human mathematical reasoning. This work investigates the transition from novice to expert problem solver [Chi81, Chi82, Larkin80]. The understanding and learning models proposed in **Physics 101** are particularly relevant for modelling the performance of humans with strong mathematical abilities when they are learning concepts in a new mathematically-based domain, such as physics. Developing a deep understanding in many domains is predicated on the ability to comprehend the constraints inherent in mathematical formalisms that model the domain.

Intelligent Computer-Aided Instruction

The application of techniques developed in artificial intelligence to the educational process is a promising endeavor [Sleeman82, Wenger87]. **Physics 101** is relevant to intelligent computer-aided instruction in two ways [Shavlik87a]. One, a model of a learning mechanism is proposed. This model predicts which types of sample problem solutions are difficult to understand and to generalize. The results of this work can be used to judge the pedagogical usefulness of various sample solutions. Two, the data structures and algorithms in the model's computer implementation might form the foundations of practical intelligent tutors for mathematically-based domains.

1.4. The BAGGER System

The **Physics 101** system provides one solution to the problem of generalizing the structure of explanations. However it is based on the semantics of mathematics and, hence, is not relevant to all domains. A second, domain-independent, approach to the issue of generalizing structure has also been proposed [Shavlik87c] and investigated in the **BAGGER** system [Shavlik87d]. In this approach, the *form* of the explanation motivates generalizing its structure.

In **BAGGER**, observations of repeated application of a rule or operator indicate that generalizing the number of rules in the explanation may be appropriate. However, alone this is insufficient. To be conducive to number generalization there must be a certain recursive structural pattern. That is, each application must achieve preconditions for the next. For example, consider stacking blocks. The same sort of repositioning of blocks occurs repeatedly, each building on the last. In **BAGGER**, the vocabulary of predicate calculus is adopted to investigate this notion of structural recursion. The desired form of structural recursion is manifested as repeated application of an inference rule in such a manner that a portion of each consequent is used to satisfy some of the antecedents of the next application.

Once it chooses a rule on which to focus attention, the system determines how an *arbitrary* number of instantiations of this rule can be concatenated together. This indefinite-length collection of rules is conceptually merged into the explanation, replacing the specific-length collection of rules, and an extension of a standard explanation-based algorithm produces a new rule from the augmented explanation. A crucial shift in representation allows the new rule to be constructed.

Rules produced by **BAGGER** have the important property that their preconditions are expressed in terms of the initial state — they do not depend on the situations implied by intermediate applications of the focus rule. This means that the results of multiple applications of the rule are determined by reasoning only about the current situation. There is no need to apply the focus rule successively, each time checking whether the preconditions for the next application are satisfied. This leads to a substantial gain in efficiency.

An empirical demonstration of the value of generalizing number in particular and explanation-based learning in general has been performed. Three different problem solvers are compared. Two of them learn: **BAGGER** and an implementation of the **EGGS** explanation-based

generalization algorithm. The third performs no learning. **BAGGER** outperforms both of the other systems.

The ideas developed in the **BAGGER** system appear applicable to the problem of automatic programming. The concise, unambiguous, and formalizable nature of computer languages makes this field especially conducive to the advantages of an explanation-based approach to machine learning [Hill87, Frieditis86, Steier87]. A promising approach to automatic programming is to first explain how a specific problem can be solved, then produce a general algorithm by analyzing the solution in an explanation-based fashion. The iterative and recursive nature of programming mean that many of the issues in generalizing number also occur in automatic programming.

1.5. Notational Conventions

Several notational conventions are followed in the text, tables, and figures of this thesis. Predicate calculus notation is used — $f(?x ?y)$ — rather than LISP notation — $(f ?x ?y)$. Leading question marks indicate predicate calculus variables. To enhance readability, arithmetic and equality terms are presented using infix notation (e.g., $?x = ?y$). Unlike in the programming language PROLOG, the case of names is indicative of nothing. Unless explicitly scoped, all variables are universally quantified. In figures representing proof trees, arrows run from the antecedent of a rule to its consequent. When there are multiple antecedents to a rule, they are implicitly conjunctively joined.

1.6. Thesis Organization

The remainder of this thesis can be divided into two parts. Chapters 2 through 7 describe the **Physics 101** system, while chapters 3 through 11 present and analyze the **BAGGER** system. These two systems are largely independent and their descriptions can be read independently.

Chapter 2 presents a more detailed introduction to **Physics 101**, a complete problem-solving system that learns. Chapters 3 and 4 describe the process of explanation construction and generalization, respectively. The use of acquired concepts is discussed in chapter 5, where the system's problem solver is described. Throughout these chapters an example involving the acquisition of the concept of momentum conservation is used to illustrate the operation of the system. Chapter 6 provides the details of the system's major algorithms. Finally, chapter 7

contains the details of several **Physics 101** examples, which further illustrate the system's operation.

Chapter 8 provides a lengthier introduction to the **BAGGER** system and chapter 9 presents the **BAGGER** generalization algorithm. Following that, chapter 10 contains a detailed presentation of several examples, from a number of domains, addressed by the system. These examples illustrate the strengths and weaknesses of the **BAGGER** approach to learning. An empirical analysis of explanation-based learning is described next. In chapter 11, the performance of **BAGGER** is compared to an implementation of a standard explanation-based generalization algorithm and to a problem-solving system that does not learn. In addition to investigating the value of generalizing the structure of explanations, these experiments address a number of other issues in explanation-based learning. Information relevant to making decisions when designing an explanation-based learning system is reported. Two appendices follow which contain related information. Appendix A describes all of the rules used in the **BAGGER** examples, while appendix B contains additional statistics gathered in the empirical analysis of explanation-based learning.

The thesis is concluded in chapter 12. In this chapter the major contributions of this research are reviewed. Next, related approaches to generalizing the structure of explanations are described and compared to this work. Several open research issues are then described and approaches to their solution are proposed.

Chapter 2

Overview of the Physics 101 System

If in other sciences we should arrive at certainty without doubt and truth without error, it behooves us to place the foundations of knowledge in mathematics.

Roger Bacon

All the mathematical sciences are founded on relations between physical laws and laws of numbers, so that the aim of exact science is to reduce the problems of nature to the determination of quantities by operations with numbers.

James Clerk Maxwell

Mathematics is the queen of the sciences.

Carl Friedrich Gauss

Mathematically-based domains present several unique challenges and opportunities for machine learning. Many scientific and technical domains - physics, electronics, chemistry, economics - share the common formalism of mathematics, and much of the reasoning in these fields involves understanding the constraints inherent in mathematical descriptions. Mathematical models of real-world situations are constructed, and these mathematical abstractions are used to predict the behavior of the domain being modelled. Hence, solving quantitative problems in these domains requires a competence in symbolic mathematical

manipulation. Furthermore, since mathematics is the underlying formal language, many important domain concepts can be adequately captured only through a mathematical specification. Thus, concept learning in these domains is also rooted in mathematics.

The research reported here focusses on learning new concepts from examples in mathematically-oriented domains, using the explanation-based learning paradigm. A computer system, **Physics 101**, has been developed that embodies the theories of learning and problem solving developed. The system is fully implemented and its performance is offered as an illustration of a number of theoretical points. It is intended to model, in a psychologically-plausible manner, the acquisition of concepts taught in a first-semester college physics course [Shavlik85b, Shavlik86b, Shavlik87e] — hence, the name **Physics 101**. The model assumes competence in mathematics at the level of someone who has completed a semester of calculus.

Because explanation-based learning requires extensive domain knowledge, it clearly is not appropriate for all learning in a new domain. However, it may be useful even in early learning if the new domain relies heavily upon a domain for which the novice does have substantial knowledge. Because mathematics underlies many other domains, a novice with some mathematical sophistication may be able to make use of explanation-based techniques without extensive knowledge of the new domain.

The focus of the implementation is not physics *per se*. Very little knowledge about physics is in the system. In fact, all of its initial physics knowledge is contained in a half-dozen or so formulae, which are listed later in this chapter. None of the system's algorithms utilize knowledge about physics, except that which is captured in these initial physics formulae. Rather, the focus is on mathematical reasoning, and the domain of physics is used as a testbed, since it is an elegant domain that stresses the use of complicated mathematics.

There are three significant results of this research. The first is the notion of *obstacles* and their *cancellation*. Analyzing the cancellation of these obstacles focusses the system's attention and guides learning. The second is a solution to the problem of generalizing the structure of explanations. Third, an approach to the *operationality/generality* problem [DeJong86, Keller87b, Mitchell86, Segre87b, Shavlik87e] in explanation-based learning is presented. The notion of a *special case* is advanced to achieve operationality without sacrificing any generality of the concept.

2.1. The Learning Model

The model of learning used in the **Physics 101** system is inspired by intuitions concerning the importance of concrete experiences when acquiring abstract concepts. In complex domains like physics, people seem to understand general rules best if they are accompanied by illustrative examples. A large part of most physics texts is taken up by examples and exercises. Indeed, there is psychological evidence that a person who discovers a rule from examples learns it better than one who is taught the rule directly [Egan74] and that illustrative examples provide an important reinforcement to general rules [Bransford76, Gick83].

Figure 2.1 illustrates the student model reflected in **Physics 101**. After a physical situation is described and a problem is posed¹, the an attempt to solve the problem is made. Focus in this research is on the process of learning during a successful solution; particularly on learning from a teacher's example. When the student cannot solve a problem, he requests a solution from his instructor. The solution provided must then be verified; additional details are requested when steps in the teacher's solution cannot be understood. The process by which the student understands an example is divided into two phases. First, using his current knowledge about mathematics and physics, the student verifies that the solution is valid. At the end of this

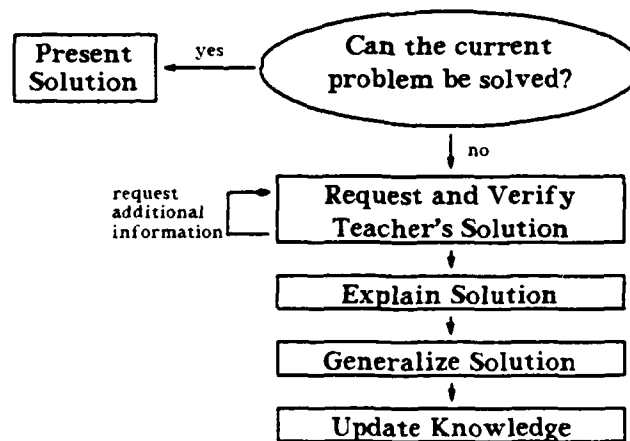


Figure 2.1 An Overview of the Learning Model

¹ Problems are expressed in the language of mathematics, not in ordinary English. The important problem of translating "word problems" into mathematical descriptions is not addressed. However, it has been addressed in the work of others [Bundy79, Novak76, Palmer83].

phase the student knows that his instructor's solution solves the current problem, but he does not have any detailed understanding of *why* his teacher chose these steps to solve the problem. During the second phase of understanding, the student determines a reason for the structure of each expression in the teacher's solution. Especially important is understanding new formulae encountered in the solution. After this phase the student has a firm understanding of how and why this solution solved the problem at hand. In the third and final phase he is able to profitably generalize any new principles that were used in the solution process, thereby increasing his knowledge of classical physics.

2.2. Some Terminology

Before the concept of an obstacle can be motivated it is necessary to introduce some terminology. The following definitions are used in **Physics 101**:

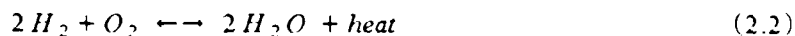
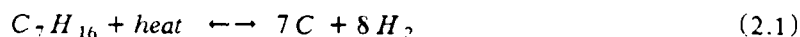
- (1) A *problem* is a set of variables, with specifications of some of the properties of these variables, together with a mathematical expression (called the *unknown*) for which the value of some property is desired. (A single, isolated variable is the simplest form of an unknown.) The properties of expressions can include such things as their value at specific points or their dependence on some other variable, such as time.
- (2) An *unacceptable* variable is one whose value for the desired property of the unknown is also not known. For example, if it is desired to know the value of the variable X at time t , and the value of Y at time t is not known, then Y is an unacceptable variable in this problem.
- (3) An *equation schema* is an inference rule which specifies a conjunction of antecedents (the preconditions) and a consequent (an equation). The equation can be used to rewrite terms in some expression. For example, the precondition of the equation $\frac{x}{x} = 1$ is that x be non-zero.
- (4) *Problem-solving schemata* describe strategies used to solve problems. They guide the problem solver in its task of applying equation schemata in order to solve the problem.
- (5) *Background knowledge* consists of equation schemata describing general mathematical laws (e.g., equations of algebra and calculus), equation schemata describing properties of the

domain at hand (e.g., *force = mass acceleration*), problem-solving schemata, and miscellaneous inference rules. The domain-specific equations specify the variables known to the system (e.g. force, mass, and acceleration) that are not mentioned in the problem description.

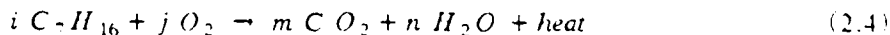
- (6) A *primary obstacle set* for a problem is a group of unacceptable variables which, if the values of their specified properties were known, would permit the system to solve the problem (that is, determine the value of the unknown's property). The first obstacle set in the problem contains those variables in the initial expression that are unacceptable. If this set is empty, the problem is solved. As obstacles from the set are assigned values for the requested property, they are eliminated from the primary obstacle set.
- (7) The *problem-solving task* consists of rewriting obstacles using equation schemata, under the guidance of solution schemata, until the primary obstacle set is empty.

2.3. Obstacles

Reasoning about obstacles plays a major role in the **Physics 101** system. Consider how the following informal example is cast in terms of obstacles. Assume the background knowledge includes reactions for forming heptane, carbon dioxide, and water:



Assume one is told that 3 moles of heptane burns completely to form carbon dioxide and water (equation 2.4, with $i = 3$).



The problem is to determine how much oxygen is consumed in the reaction. In this case, the initial primary obstacle set is $\{j\}$, the designated unknown. Using rules 2.2 and 2.3 one can see that if both m and n are known, j would be determined, as $j = m + \frac{n}{2}$. Thus, $\{m, n\}$ also

forms a primary obstacle set. Finally, rule 2.1 can be used to rewrite m and n in terms of i : $m = 7i$, $n = 8i$. Since the value of i is known, the primary obstacle set is now empty and the problem is solved.

This solution is quite easy. No learning is necessary or appropriate, but it illustrates the idea of a primary obstacle set.

Often the designated unknown cannot easily be re-expressed in terms of acceptable variables. A more interesting technique, and the technique focussed on, is to introduce new obstacles or regroup existing variables in such a way that, after suitable rewriting, the obstacles are *cancelled*. This type of mathematical reasoning supports many important domain concepts. A later detailed example shows **Physics 101** applying this technique to acquire the concept of conservation of momentum.

For an intuitive grasp of the technique, consider the following elementary physics problem:

A massless point particle of charge 16 esu experiences an attraction to a charged sphere whose radius is 3 cm . The particle is 7 cm from the surface of the sphere. The sphere's charge is uniformly distributed about its surface with a density of $0.2 \text{ esu} / \text{cm}^2$. What is the force experienced by the particle?

Assume the background knowledge includes the formula for the electrostatic force between point charges, formulae for the volume and surface areas of a sphere, superposition of forces, etc. There are two ways to work this problem. The first is to grind through the equations, performing a surface integral over the sphere of the force contribution of each differential area. The second is to compute the total charge on the sphere and treat it as an equivalent point charge located at the sphere's center.

The second solution is much simpler, but it is not immediately apparent from the background knowledge that it is a valid solution. Somewhat sophisticated mathematical reasoning is needed to confirm this approach. Basically, it works for two reasons. First, due to the symmetry of the situation, the net force on the particle lies on the line connecting the particle and the center of the charged sphere. Forces in all other directions are cancelled. Second, the particle is attracted to the near portion of the sphere just as much as the far portion. This is a bit counter-intuitive. Even though the front of the sphere is closer to the particle, any

differential solid angle cone from the particle through the sphere intercepts a slightly larger area on the far side than the near side. The larger size of the rear differential exactly balances the closeness advantage of the front differential.

If a student is shown the second, easier solution, and if he can convince himself that the simplifying transformation is correct, then he can learn an important new problem-solving technique. His performance will be improved for a class of conceptually similar, but possibly superficially very different problems.

The notion of obstacles is central to determining the range of applicability of the new concept. The concept's generality is dictated by the generality of the inference rules participating in obstacle cancellation. Any problem which allows the same obstacle cancellation treatment will be covered by the new concept. The trick works, for example, in a problem involving a gravitational force instead of electrostatic force - both are inverse square forces, but it would not work if the particle were *inside* the sphere as the obstacle cancellation structure is no longer supported.

This notion of a *primary obstacle set* will be illustrated in more detail later. For now it is sufficient to understand that it is a set of unacceptable variables, descended from the designated unknown, which together would allow a value to be attributed to the desired property of the designated unknown. The task in problem solving is to circumvent the need to know the properties of these obstacles. The additional notion of *secondary obstacles* will be discussed later. Note that secondary obstacles are *not* simply descendents of descendents of the designated unknown. These are also primary obstacles. Rather, the term *secondary obstacles* is used to refer to obstacles unavoidably introduced during the process of cancelling primary obstacles.

2.4. Special-Case Schemata

Acquiring the definition of a new concept is only half of the problem faced by a machine learning system. To usefully contribute to improved problem solving, the system must be able to later apply the new concept effectively. That is, the concept must be *operational* [Mostow 83].

There is a trade-off between operationality and generality of concepts. For example, suppose one has the goal of alleviating his hunger. Compare two possible schemata for satisfying this goal. One schema might be *eat-at-Chez-Jean*, which specifies that eating at an

expensive restaurant would suffice. This schema includes such preconditions as being hungry for French food, having a lot of money, wanting to eat after seven pm (the time it opens), and having three hours to spare for dinner. A second problem-solving schema is *eat-at-restaurant*. It applies to many different restaurants. Preconditions are specified as constraints among potential bindings for schema variables rather than actual values. Thus, instead of specifying French food, seven pm, and three hours, it states the mutual interdependence among the restaurant chosen, quality of food, type of food, and price.

The second schema is more general than the first. One can apply the second at *McDonald's*, *Pop's Malt Shop*, and *China Inn*, as well as *Chez Jean*. Although subsumed by the second, the first schema is more efficient to apply. If one knows that applying the *Chez-Jean* schema leads to a satisfactory solution, one need not solve the potentially difficult constraint satisfaction problem posed by the precondition structure of the more general restaurant schema.

What underlies this generality? It is achieved by calling the problem solver to create and select among consistent sets of alternative variable bindings needed to satisfy the schema's preconditions. This problem-solving must be done at the time the schema is applied. It cannot be "pre-compiled" since binding any of the schema variables reduces the schema's applicability. *Internal problem solving* refers to this kind of "application-time" problem solving. Performing internal problem solving is not free. It requires resources of time and effort. Thus, the more internal problem solving, the greater the potential generality, but the less operational the resulting schema.

In learning a new schema, it would seem that the system must weigh operability against generality. However, this need not be the case. By acquiring several schemata, operability can be preserved without sacrificing generality. In addition to the general schema, specializations of the concept, called *special-case schemata*, are formed and stored. In section 4.4, a method for forming special cases, which guarantees improved operability and yields schemata with potentially high utility, is presented.

2.5. Initial Knowledge of the System

Physics 101 possesses a large number of mathematical problem-solving techniques. For example, it can symbolically integrate expressions, cancel variables, perform arithmetic, and replace terms by utilizing known formulae. Figure 2.2 contains a portion of the system's

hierarchy of calculation techniques. A calculation step may either rearrange the entities in the current expression, simplify the current expression, or replace terms in the expression by substituting formulae. Rearrangement involves such things as moving constants into and out of integrals and derivatives. Simplification involves algebraic cancellation, numerical calculation, and the solving of calculus. The formulae that may be used to replace variables existing in an expression are determined by the domain being investigated. Chapter 5 contains further details on the different strategies for substituting formulae.

Figure 2.3 contains some of the general mathematical rewrite rules known to the system. Note some formulae belong to more than one category, depending on use. (Terms beginning with a question mark are universally quantified variables.)

Figure 2.4 contains the initial physics formulae provided to the system, along with the conditions on their applicability. The first two formulae in figure 2.4 define the physical concepts of velocity (V) and acceleration (A). An object's velocity is the time rate of change of

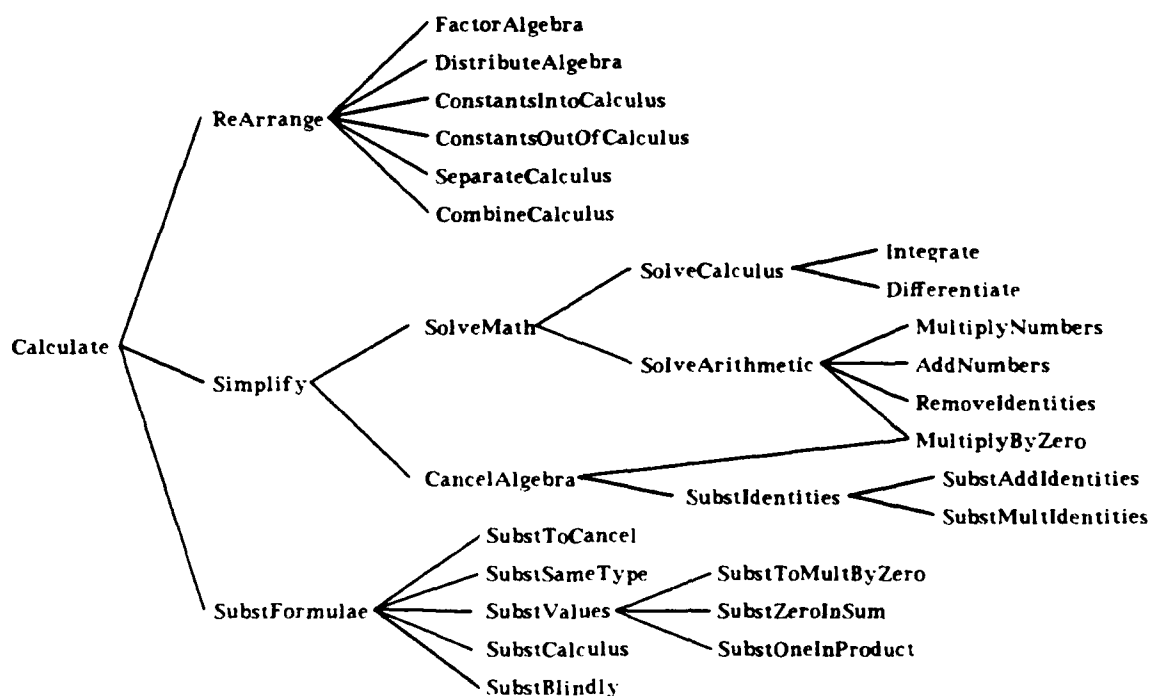


Figure 2.2 Some of the System's Mathematical Techniques

$$?expression - ?expression = 0$$

Problem-Solving Schema: SubstAddIdentities

$$?expression + 0 = ?expression$$

Problem-Solving Schema: RemoveIdentities

$$?expression / ?expression = 1$$

Problem-Solving Schema: SubstMultIdentities

Preconditions: NOT(ZeroValued(?expression))

$$1 * ?expression = ?expression$$

Problem-Solving Schema: RemoveIdentities

$$\int (?independent * ?expression) d?x = ?independent \int ?expression d?x$$

Problem-Solving Schema: ConstantsOutOfCalculus

Preconditions: IndependentOf(?independent, ?x)

$$\int (?expression_1 + ?expression_2) d?x = \frac{d}{d?x} ?expression_1 + \frac{d}{d?x} ?expression_2$$

Problem-Solving Schema: SeparateCalculus

$$\int ?expression_1 d?x + \int ?expression_2 d?x = \int (?expression_1 + ?expression_2) d?x$$

Problem-Solving Schema: CombineCalculus

$$\frac{d}{d?x} ?expression^n = ?n ?expression \frac{d}{d?x} ?expression^{n-1}$$

Problem-Solving Schema: Differentiate

Preconditions: Number(?n)

$$\int ?expression d?x = ?independent * ?x + constant$$

Problem-Solving Schema: Integrate

Preconditions: IndependentOf(?independent, ?x)

Figure 2.3 Sample Mathematical Rewrite Rules

$$V_{?i,?c}(t) = \frac{d}{dt} X_{?i,?c}(t)$$

Preconditions: $\text{Member}(?i, \text{ObjectsInWorld}) \wedge \text{IsaComponent}(?c)$

$$A_{?i,?c}(t) = \frac{d}{dt} V_{?i,?c}(t)$$

Preconditions: $\text{Member}(?i, \text{ObjectsInWorld}) \wedge \text{IsaComponent}(?c)$

$$F_{net,?i,?c}(?t) = M_{?i} * A_{?i,?c}(?t)$$

Preconditions: $\text{Member}(?i, \text{ObjectsInWorld}) \wedge \text{IsaComponent}(?c) \wedge \text{IsaTime}(?t)$

$$F_{net,?i,?c}(?t) = F_{ext,?i,?c}(?t) + F_{int,?i,?c}(?t)$$

Preconditions: $\text{Member}(?i, \text{ObjectsInWorld}) \wedge \text{IsaComponent}(?c) \wedge \text{IsaTime}(?t)$

$$F_{int,?i,?c}(?t) = \sum_{\substack{j \in \text{ObjectsInWorld} \\ j \neq ?i}} F_{j,?i,?c}(?t)$$

Preconditions: $\text{Member}(?i, \text{ObjectsInWorld}) \wedge \text{IsaComponent}(?c) \wedge \text{IsaTime}(?t)$

$$F_{?j,?i,?c}(?t) = -F_{?i,?j,?c}(?t)$$

Preconditions: $\text{Member}(?i, \text{ObjectsInWorld}) \wedge \text{Member}(?j, \text{ObjectsInWorld}) \wedge$
 $?i \neq ?j \wedge \text{IsaComponent}(?c) \wedge \text{IsaTime}(?t)$

Figure 2.4 The Initial Physics Formulae of the System

its position (X) and its acceleration is the time rate of change of its velocity. Newton's second and third laws are also included. (Newton's first law need not be included because it is a special case of his second law.) The second law states that the net force (F) on an object equals its mass (M) times its acceleration (A). The net force is decomposed into two components: the external force (F_{ext}) and the internal force (F_{int}). External forces result from any external fields (e.g., gravity) that act upon objects. Object $?i$'s internal force is the sum of the forces the

other objects in the world² exert on object $?i$. These *inter-object* forces are constrained by Newton's third law, which says that every action has an equal and opposite reaction.

Position, velocity, acceleration, and force are spatial *vectors*. Hence one of their arguments ($?c$) indicates which vector component is being discussed (x , y , or z). All of these physical variables are functions of time. Mass, however, is a time-independent scalar. It is only indexed by the physical object whose mass is being specified.

When a problem is described, the number of objects in the world is specified. At this time, summations (Σ 's) and products (Π 's) in the known formulae are expanded. For instance, if there are three objects in a world, the second from last equation in figure 2.4 becomes:

$$F_{int, ?i, ?c}(?t) = F_{?j, 1, ?i, ?c}(?t) + F_{?j, 2, ?i, ?c}(?t)$$

Preconditions:

$$\text{Permutation}(\{?i, ?j, 1, ?j, 2\}, 1\text{ObjectsInWorld}) \wedge \text{IsaComponent}(?c) \wedge \text{IsaTime}(?t)$$

Expanded equations are produced because of their greater psychological-plausibility as a model of learning by a college freshman.

World-specific equations can also be provided when a new problem is posed. For example, it may be stated that the external force on object k is $M_k g X_{k,y}$. While formulae of these types may be used in solutions to specific examples, they could unnecessarily constrain generalization, since they do not hold in *all* worlds. The handling of expanded and problem-specific formulae during the generalization process is discussed in chapter 4.

2.6. Summary

This chapter provides an overview of the **Physics 101** system, a system that performs explanation-based learning in mathematically-oriented domains. It presents the model of learning employed, defines terms used in the next few chapters, introduces the important concept of an obstacle, introduces the concept of special-case schemata, and presents the initial knowledge of the system.

² The term *world* is used to refer to physical systems and situations. The term *system* is reserved for referring to computer programs.

The following five chapters elaborate these issues. The next chapter discusses the explanation of mathematical calculations. Chapter 4 describes the way in which these explanations are generalized, further motivates the need for generalizing the structure of explanations, and discusses how special cases are constructed. One important reason for learning is to enhance problem-solving performance [Simon83]. Chapter 5 contains a discussion about the system's problem solver, including how **Physics 101** uses the new schemata it acquires. The following chapter, chapter 6, contains the details of the system's major algorithms. Finally, chapter 7 presents additional examples processed by the system and discusses the issues addressed by these examples.

Chapter 3

Building Explanations in Physics 101

In explanation-based learning, the solution to a specific sample problem is generalized and the result saved, in the hopes of being applicable to future problems. This chapter addresses the construction of explanations in mathematically-based domains, while the next addresses the generalization of these explanations. The focus is on understanding solutions, presented by a teacher, to problems the system could not solve on its own.¹ During this understanding process, gaps in the teacher-provided solution may need to be filled. The teacher must provide enough guidance so that the system's limited problem solver can successfully solve the problem. If the provided solution can be sufficiently-well understood, a new schema may result, and the next time the system faces a problem related to the current one it may be able to solve it without the need for external assistance.

¹ Although the emphasis is on teacher-provided solutions, much of the discussion in this chapter also applies to the explanation of the system's own problem solving. It may take a substantial amount of work for **Physics 101** to solve a problem, even without needing external help, and this effort can be reduced in the future by creating a new schema from the analysis of its labors. See chapter 11 for further discussion on the issue of a system learning from its own problem solving.

Understanding a teacher-provided solution involves two phases. First, the system attempts to verify that each of the instructor's solution steps mathematically follows. If successful, in the second phase the mathematical reasoning component of **Physics 101** builds an explanation of *why* the solution works. A sample collision problem illustrates these two phases.

3.1. A Sample Problem

In the one-dimensional problem shown in figure 3.1, there are three balls moving in free space, without the influence of any external forces. (Nothing is specified about the forces between the balls. Besides their mutual gravitational attraction, there could be a long-range electrical interaction and a very complicated interaction during the collision.) In the initial state (**state A**) the first ball is moving toward the other two, which are stationary. Some time later (**state B**) the second and third balls are recoiling from the resulting collision. The task is to determine the velocity of the first ball after the collision.

Physics 101 cannot solve this problem with only the schemata shown in figure 2.4. Initially, the formula $V = \frac{d}{dt}X$ is tried, which leads nowhere, then the solution steps presented in figure 3.2 are attempted. The system's problem solver is incomplete, for reasons detailed in chapter 5. One source of incompleteness occurs because the problem solver never performs two

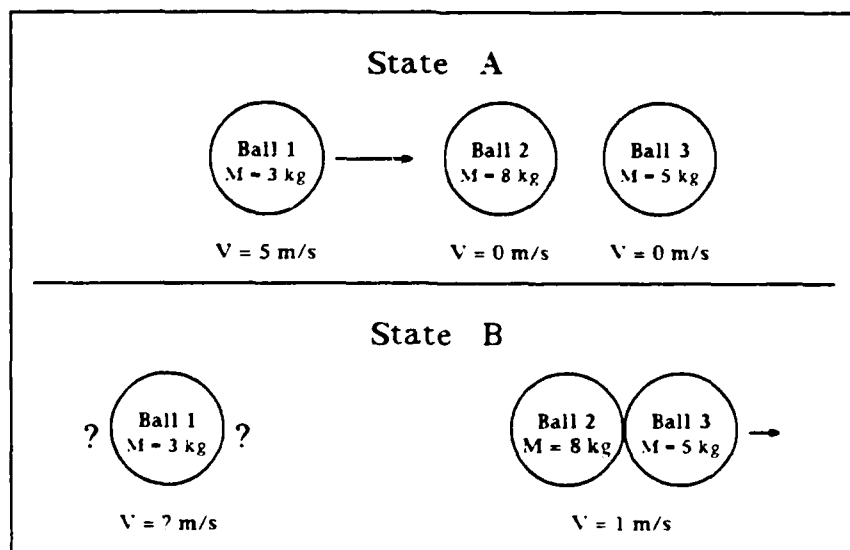


Figure 3.1 A Three-Body Collision Problem

$$V_{1,x}(t)$$

$$1 \text{ SubstSameType} = \int A_{1,x}(t) dt$$

$$2 \text{ SubstBlindly} = \int \frac{F_{\text{net},1,x}(t)}{M_1} dt$$

Figure 3.2 The Failed Solution Attempt

unmotivated ("blind") substitutions consecutively. The problem solver possesses no motivated strategy to lead it past line 2 in figure 3.2, and asks for a solution from its teacher.

The teacher's solution to the collision problem can be seen in figure 3.3. Without being explicitly stated, the principle of conservation of momentum is being invoked, as the momentum ($M \times V$) of the balls at two different times is equated. This equation is not a variation of any formula known to the system (figure 2.4). A physically-consistent mathematical derivation is needed if **Physics 101** is to accept the solution provided.

3.2. Verifying a Teacher's Solution

In order to accept a teacher's answer, the system must verify each of the steps in the solution. Besides being mathematically correct, the calculations must be consistent with its domain-specific knowledge. To be valid, each of the solution steps must be assigned to one of the following four classifications.

$$M_1 V_{1,x}(A) + M_2 V_{2,x}(A) + M_3 V_{3,x}(A) = M_1 V_{1,x}(B) + M_2 V_{2,x}(B) + M_3 V_{3,x}(B)$$

$$(3 \text{ kg})(-5 \frac{\pi}{s}) = 8 \text{ kg } V_{1,x}(B) + (3 \text{ kg})(1 \frac{\pi}{s}) + (5 \text{ kg})(1 \frac{\pi}{s})$$

$$-15 \frac{\text{kg} \pi}{s} = 8 \text{ kg } V_{1,x}(B) + 8 \frac{\text{kg} \pi}{s}$$

$$V_{1,x}(B) = -2.88 \frac{\pi}{s}$$

Figure 3.3 The Teacher's Solution

- (1) Instantiation of a known formula: *force = mass \times acceleration* is an example of this type.
- (2) Definition of a new variable in order to shorten later expressions: *resistance = voltage / current* would fall in this category.
- (3) Rearrangement of a previously-used formula. These equations are mathematical variants of previous steps. The replacement of variables by their values also falls into this category.
- (4) Statement of an unknown relationship among known variables. These steps require full justification, which the system performs symbolically by reasoning about algebra and calculus. Only the steps in this category are candidates for generalization.

Physics 101 possesses several methods for verifying equations falling into category 4. Two are suggested when the two sides of an equation only differ as to the time at which they are evaluated (a condition satisfied by the initial equation in figure 3.3). One method this suggests is to determine if the common form underlying each side of the equation is constant with respect to time. This can be done by seeing if its derivative is zero. The second suggested method is to determine how the underlying form explicitly depends on time. If time can be made explicit, it is easy to see if it is valid to equate the expression at two different times. This method can handle more situations than the first, and is the one used by **Physics 101** to understand the teacher's solution.

Figure 3.4 illustrates three possible forms of the underlying time-dependent expression. The expression could be periodic, and hence could be equated at times separated by some number

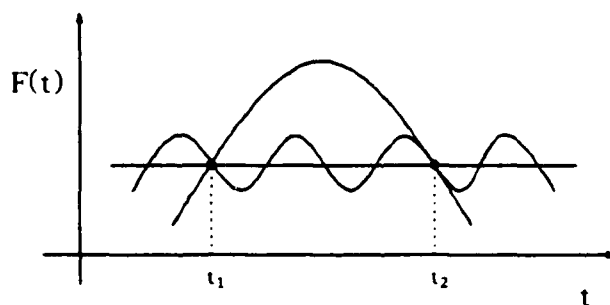


Figure 3.4 Equating an Expression at Two Times

of periods. Alternatively, the expression could be parabolic. Here there would be some quadratic relationship between times where it is valid to equate the expression. A third possibility is that the expression is constant with respect to time. In this last case, it is valid to equate the expression at *any* two times.

Once the system selects a method for verifying a new equation, it must perform the mathematics necessary to determine any additional information required by the method. For example, it may need to determine the derivative of an expression or legally eliminate all the terms whose time-dependence is not known. In this phase, **Physics 101** is a mathematical problem solver.

The actual calculations of the system appear in figure 3.5. The goal is to convert, via a series of equality-preserving transformations, the top expression into an equivalent expression whose time dependence is explicit. Once this is done, the system can determine if figure 3.3's equation 1 is valid. (The top expression in figure 3.5 is called the *left-hand side* of the calculation, while the other expressions are termed *right-hand sides*.)

Three qualitatively different strategies are used during various stages of problem solving. The first strategy is to apply operators that will lead to definite progress toward the goal. Attention is focussed by this strategy as long as some operator in this class is applicable. When clear progress cannot be achieved, the problem solver must decide how best to proceed. The second strategy is then invoked to select operators that preserve important characteristics of the current problem. These operators are likely to keep the problem solver from diverging sharply from the goal while possibly enabling the application of operators by the first strategy. When the problem solver can follow neither of the first two strategies, the third strategy of arbitrarily applying legal operators is used. This problem-solving model is described more fully in chapter 5. There, several mathematical calculation operators used under each strategy are described.

The annotations to the left of the expressions in figure 3.5 are produced by the system. These annotations indicate which of **Physics 101**'s problem-solving schemata (figure 2.2) is used to perform each calculation step. In the first step, the formulae substitutions are not chosen in

$$\begin{aligned}
& M_1 V_{1,x}(t) + M_2 V_{2,x}(t) + M_3 V_{3,x}(t) \\
1 \quad \text{SubstSameType} &= M_1 \int A_{1,x}(t) dt + M_2 \int A_{2,x}(t) dt + M_3 \int A_{3,x}(t) dt \\
2 \quad \text{SubstToCancel} &= M_1 \int \frac{F_{net,1,x}(t)}{M_1} dt + M_2 \int \frac{F_{net,2,x}(t)}{M_2} dt + M_3 \int \frac{F_{net,3,x}(t)}{M_3} dt \\
3 \quad \text{ConstsOutCalculus} &= \frac{M_1}{M_1} \int F_{net,1,x}(t) dt + \frac{M_2}{M_2} \int F_{net,2,x}(t) dt + \frac{M_3}{M_3} \int F_{net,3,x}(t) dt \\
4 \quad \text{SubMultIdentities} &= 1 \int F_{net,1,x}(t) dt + 1 \int F_{net,2,x}(t) dt + 1 \int F_{net,3,x}(t) dt \\
5 \quad \text{RemoveIdentities} &= \int F_{net,1,x}(t) dt + \int F_{net,2,x}(t) dt + \int F_{net,3,x}(t) dt \\
6 \quad \text{SubstSameType} &= \int (F_{ext,1,x}(t) + F_{int,1,x}(t)) dt + \int (F_{ext,2,x}(t) + F_{int,2,x}(t)) dt \\
& \quad + \int (F_{ext,3,x}(t) + F_{int,3,x}(t)) dt \\
7 \quad \text{SubstSameType} &= \int (F_{ext,1,x}(t) + F_{2,1,x}(t) + F_{3,1,x}(t)) dt + \int (F_{ext,2,x}(t) + F_{1,2,x}(t) + F_{3,2,x}(t)) dt \\
& \quad + \int (F_{ext,3,x}(t) + F_{1,3,x}(t) + F_{2,3,x}(t)) dt \\
8 \quad \text{SubstToCancel} &= \int (F_{ext,1,x}(t) + F_{2,1,x}(t) + F_{3,1,x}(t)) dt + \int (F_{ext,2,x}(t) - F_{2,1,x}(t) + F_{3,2,x}(t)) dt \\
& \quad + \int (F_{ext,3,x}(t) - F_{3,1,x}(t) - F_{3,2,x}(t)) dt \\
9 \quad \text{CombineCalculus} &= \int (F_{ext,1,x}(t) + F_{2,1,x}(t) + F_{3,1,x}(t) + F_{ext,2,x}(t) - F_{2,1,x}(t) + F_{3,2,x}(t) \\
& \quad + F_{ext,3,x}(t) - F_{3,1,x}(t) - F_{3,2,x}(t)) dt \\
10 \quad \text{SubAddIdentities} &= \int (F_{ext,1,x}(t) + 0 \frac{kg \cdot m}{s^2} + 0 \frac{kg \cdot m}{s^2} + F_{ext,2,x}(t) + 0 \frac{kg \cdot m}{s^2} + F_{ext,3,x}(t)) dt \\
11 \quad \text{AddNumbers} &= \int (F_{ext,1,x}(t) + F_{ext,2,x}(t) + F_{ext,3,x}(t) + 0 \frac{kg \cdot m}{s^2}) dt \\
12 \quad \text{RemoveIdentities} &= \int (F_{ext,1,x}(t) + F_{ext,2,x}(t) + F_{ext,3,x}(t)) dt \\
13 \quad \text{SubstValues} &= \int (0 \frac{kg \cdot m}{s^2} + 0 \frac{kg \cdot m}{s^2} + 0 \frac{kg \cdot m}{s^2}) dt \\
14 \quad \text{AddNumbers} &= \int 0 \frac{kg \cdot m}{s^2} dt \\
15 \quad \text{Integrate} &= constant_1
\end{aligned}$$

Figure 3.5 Verifying the First Equation in Figure 3.3

support of a variable cancellation.² In the next step, the formulae substitutions are chosen because the mass terms can be cancelled. Before this cancellation can take place, however, the cancelling terms must be brought together. The calculation continues in a like manner until all the unknown variables are eliminated. Then the known values are substituted and the ensuing arithmetic and calculus is solved. Since the initial expression is constant, it can be equated at any two times. Equation 1 of figure 3.5 is valid.

3.3. Explaining Solutions

At this point, the system has ascertained that its teacher's use of a new equation is indeed valid. Figure 3.5 can be viewed as an explanation structure. Underlying the calculation are a large number of equation schemata, along with their associated preconditions. These schemata justify the transformations from one line to the next. Between each pair of consecutive expressions are the equation schemata (and their preconditions) used to rewrite one expression into the next. This is illustrated in figure 3.6. The left side of this figure shows the general

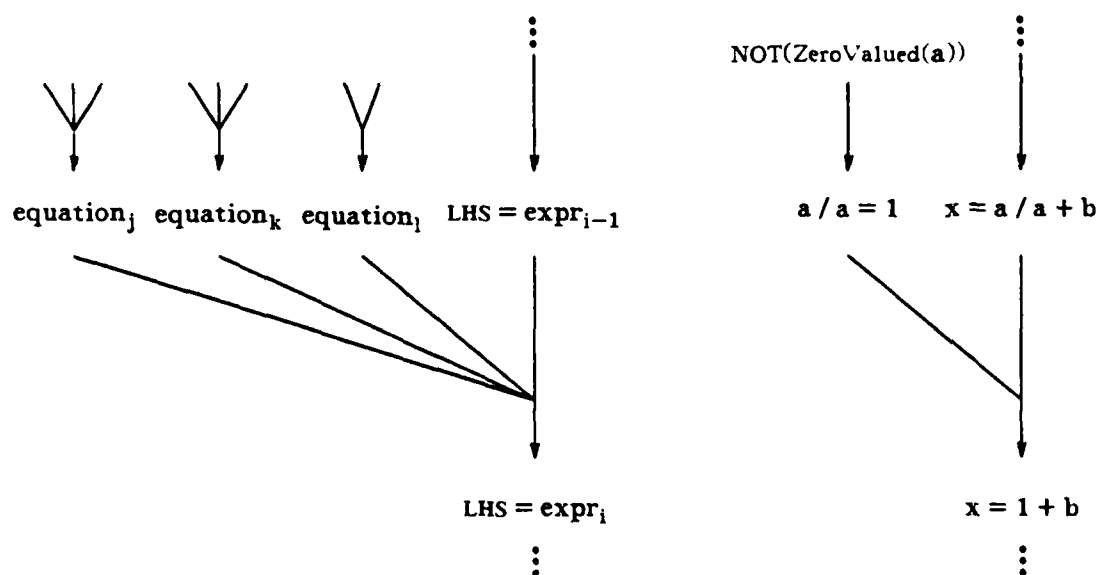


Figure 3.6 The Underlying Structure of a Calculation

² Initially, the system chooses to replace the velocities by the derivative of the positions. This leads nowhere and the system backtracks. No other backtracking occurs during the calculation of figure 3.5. The system is guided by the goal of cancelling variables, which greatly reduces the amount of unnecessary substitutions during problem solving.

structure existing between consecutive expressions, while the right half presents a simple, concrete example.

However, although figure 3.5 constitutes a perfectly acceptable explanation of the solution to the specific example of figure 3.1, the underlying explanation does not directly suffice to produce the proper general concept. Applying a standard explanation-based generalization algorithm does produce a generalization of the specific solution (see section 4.2), and a number of attributes of the problem are generalized. For example, the result does not only apply to colliding balls - it applies to situations involving any type of physical object. Nor does the problem have to be in the x -direction, since none of the schemata used to tie the calculation together constrain the component of the variables. A third property generalized is that the external forces need not individually be zero. All that step 15 of figure 3.5 requires is that the external forces sum to zero. However, since the explanation structure is not generalized, one unfortunate aspect of the specific example remains. The new law only applies to physical situations involving *three* objects. Without that property, adding the $M V$ terms of three objects will not lead to the complete cancellation of internal forces of the objects. The preconditions of the new schema would insist on a three-object world without external forces, because only then will the sum of three momentum terms always be constant across time. Unfortunately this result is not broadly applicable. The system would need to learn separate rules when it encountered a four-object system, a five-object system, etc.

In order to produce the proper generalization to N , the system must determine a reason for including each variable in this equation. This will determine which variables are required in its general form.

In the explanation process, **Physics 101** determines how the value of the desired property of the current problem's *unknown* is obtained. As stated earlier, the problem's unknown is the expression about which the value of some property is being sought; in the sample problem, V_1 is the unknown and its value in state B is being sought. During this process, the system determines the role of each variable in the initial expression of the calculation.

During a calculation one of three things can happen to a variable:

- (1) its value can be substituted,
- (2) it can be symbolically replaced during a formulae substitution, or
- (3) it can be *cancelled*.

Understanding and generalizing variable cancellation drives **Physics 101**. The system can identify the first five of the following six types of variable cancellations:

additive identity

These are algebraic cancellations of the form $x - x = 0$. Line 10 in figure 3.5 contains two additive cancellations.

multiplicative identity

These are algebraic cancellations of the form $x / x = 1$. Line 4 in figure 3.5 involves two multiplicative cancellations.

multiplication by zero

These are cancellations that result from an expression (which may contain several variables) being multiplied by zero. None appear in figure 3.5.

integration (to a number)

This type of cancellation occurs when variables disappear during symbolic integration. When integration produces *new* variables (other than the integration constant), this calculation is viewed as a substitution involving the original terms. No cancellations of this type appear in figure 3.5.

differentiation (to a number)

This is analogous to cancellation during integration.

assumed ignorable

A term can be additively ignored because it is assumed to be approximately zero or multiplicatively ignored because it is assumed to be approximately equal to one.

One of the most important aspects of using mathematical techniques to solve real-world problems is knowing how to create a mathematically tractable problem out of a given

situation. Learning how to do this by observing successful problem solvers produce a useful approximation to a complicated problem is an area for future research. A system that uses its knowledge about valid approximations to learn approximate solutions to intractable problems is presented in [Bennett87].

3.4. Understanding Obstacles

Recall that *obstacles* are expressions appearing in a calculation but whose values are not known. *Primary obstacles* are obstacles descended from the unknown. In the momentum problem the only primary obstacles not replaced in a formula substitution are $F_{2,1}$ and $F_{3,1}$. If the value of the desired property of each of the primary obstacles were known, the value of the unknown's desired property would be specified. The system ascertains how these obstacles are eliminated from the calculation. Cancelling obstacles is seen as the essence of the solution strategy, because when all the obstacles have been cancelled the value of the unknown's desired property can be easily calculated.

Figure 3.7 illustrates the concept of primary obstacles. The goal in this sample problem is to determine the value of V_1 . Since this is not known, the problem is transformed to that of finding A_1 (for simplicity, the integral sign is ignored here). However, the value of A_1 is not known either. This leads to the substitution of $F_{net,1}$ divided by M_1 . The mass is known, but the net force is not. The net force is then decomposed into two components - a known external force and an unknown internal force. Finally, the internal force is further decomposed into its

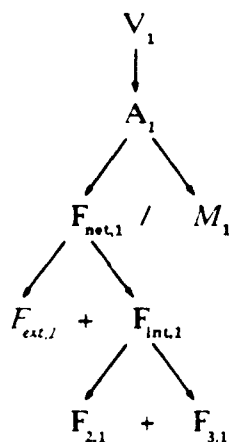


Figure 3.7 Decomposing the Unknown

constituents. These two inter-object forces are the obstacles to knowing the value of $V_{1,x}$. **Physics 101** needs to determine how the solution in figure 3.3 circumvents the need to know the value of these two variables.

Figure 3.8 contains the *cancellation graph* for the three-body collision problem. This data structure is built by the system during the understanding of the specific solution. It holds the information that explains how the specific example's obstacles are eliminated from the calculation. This information is used to guide the generalization process described in the next chapter.

To reason about a calculation, **Physics 101** must be able to distinguish different instances of the same variable. For example, the M_1 introduced in line 2 of figure 3.5 plays a different role than the M_1 appearing in the left-hand side of the equation. In the system, variables are

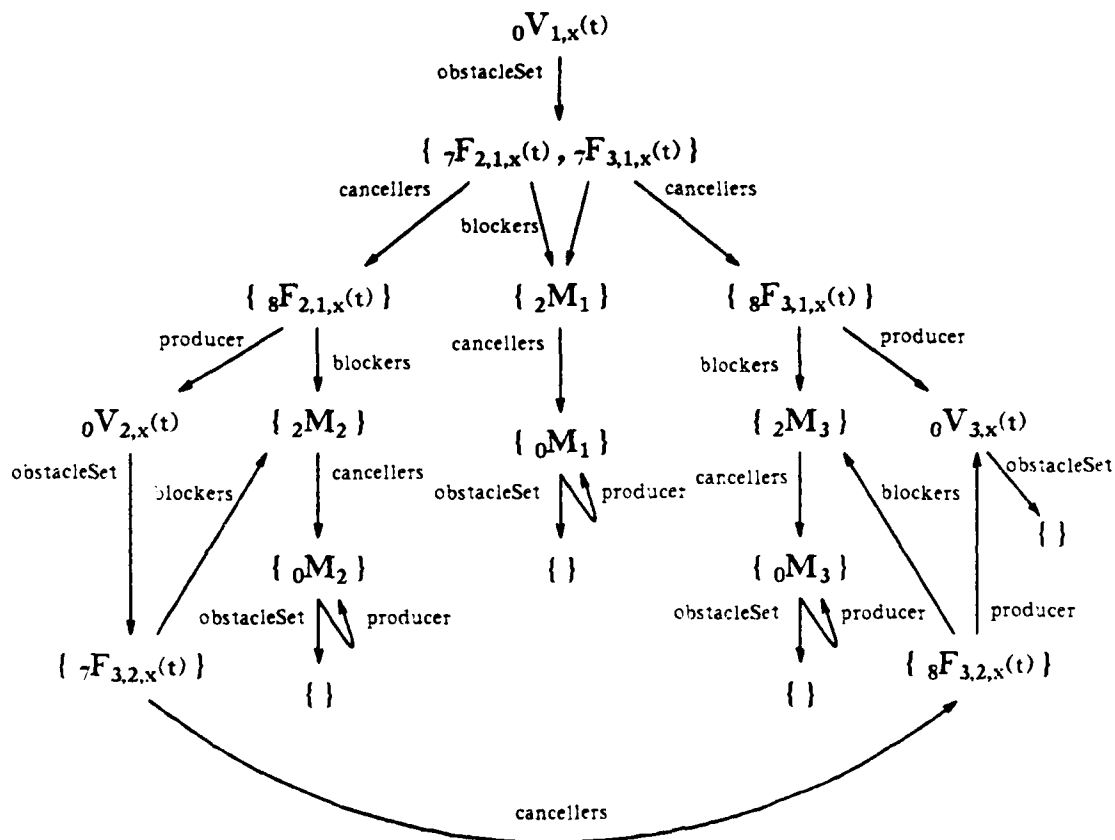


Figure 3.8 The Cancellation Graph

marked to record in which solution step they first appear. This information is recorded in the above cancellation graph (and in subsequent cancellation graphs) by the subscript preceding a variable. Variables originating in the left-hand side are prefixed with zeroes.

To understand the calculation, the system first determines that the primary obstacles $F_{2,1,x}$ and $F_{3,1,x}$ are eliminated by being *additively cancelled*. Although cancelled additively, these variables descended from a multiplicative expression ($A = \frac{F}{M}$). Hence, the system must determine how they are *additively isolated*. Multiplication by M_1 performed this task. So an explanation of the M_1 term in the left-hand side expression of figure 3.5* is obtained.

The next thing to do is to determine how the terms that additively cancel $F_{2,1,x}$ and $F_{3,1,x}$ are introduced into the calculation. $F_{2,1,x}$ is cancelled by a force descended from $V_{2,x}$. This $F_{2,1,x}$, too, must first be additively isolated. **Physics 101** discovers that the left-hand side's M_2 performs this isolation. The system now has explanations for the M_2 and the $V_{2,x}$ terms in the left-hand side. Similar reasoning determines the role of M_3 and $V_{3,x}$.

Cancellation of the primary obstacles requires the presence of additional variables on the left-hand side of the equation. These extra terms may themselves contain obstacle variables. These are called *secondary obstacles*. The system must also determine how these obstacles are eliminated from the calculation. The elimination of the secondary obstacles may in turn require the presence of additional variables in the left-hand side expression, which may introduce additional secondary obstacles. This recursion must terminate, however, as the calculation is known to have eliminated all of the unacceptable terms.

Cancelling the inter-object forces involving ball 1 introduced one secondary obstacle - $F_{3,2,x}$. This secondary obstacle was additively cancelled by a force descended from $V_{3,x}$. Cancelling this secondary obstacle produced no new obstacles.

Once the system determines how all of the obstacles in the calculation are cancelled, generalization can occur. At this time, **Physics 101** can also report any variables in the left-hand side of a calculation that are irrelevant to the determination of the value of the unknown. Those variables not visited during the construction of the cancellation graph are not necessary, even though they are present in the teacher's solution.

The following describes the categories involved in understanding obstacles and their circumvention. More details on these categories are provided in chapter 6, which presents the system's algorithms.

Obstacles

As defined above, obstacles are variables that cannot appear on the final right-hand side of the current calculation. There are two types of obstacles: primary obstacles are descended from the problem's unknown; secondary obstacles are by-products of the cancellation of obstacles.

There are several types of variables associated with obstacle elimination.

Obstacle Blockers

These are variables preventing additive or multiplicative access (depending on the manner of cancellation) to an obstacle. For example, in figure 3.5 M_1 is preventing additive access to the obstacle $F_{2,1,x}$. Blockers must first be eliminated before the obstacle can be cancelled.

Obstacle Partners

These are variables that must be present in combination with the obstacle in order for the obstacle to be cancelled.

Obstacle Cancellers

These are the variables that directly cancel an obstacle. Finally, in order to check for secondary obstacles, the *producers* of variables must be known.

Producers

The producer of a variable is the variable(s) in the left-hand side of a calculation that, through the application of a series of rewrite rules, ultimately lead to the its existence.

3.5. Summary

This chapter discusses the construction of explanations in mathematically-based domains. An approach to categorizing and then embellishing teacher-provided solutions is presented first. Equations in a teacher's solution that are seen as containing something significantly new are

explained in detail, during which the system performs some focussed problem solving. Once the system ascertains that an interesting solution step is mathematically correct, a more abstract explanation of the step is constructed. This abstract explanation is needed if **Physics 101** is to produce the proper generalization. Simply applying standard explanation-based learning techniques to an explanation based on the mathematical rewrite rules used is not sufficient. Rather, the fuller explanation is required to guide the construction of the general version of the calculation. In the fuller explanation, the manner by which the calculation eliminates the obstacles in the calculation is recorded. The goal of eliminating the general versions of these obstacles leads to the necessary restructuring of the original calculation and to a new, general concept. The generalization process is the topic of the next chapter.

Chapter 4

Generalizing Solutions in Physics 101

Once the solution to a problem is understood, it must be generalized so that it can be used to help solve similar problems in the future. In **Physics 101** this involves generalizing the structure of the specific problem's solution, as well as generalizing the constants in the specific example. This chapter presents the process by which the general version of a specific problem's solution is constructed. The reasons why the general and specific calculation differ are described. Also in this chapter the generalization produced by standard explanation-based learning algorithms is presented, the construction of special-case schemata is discussed, performance improvement after learning is empirically analyzed, and other approaches to learning in mathematical domains are described. Chapter 6 presents the generalization algorithm in more detail, while additional examples are presented in chapter 7.

4.1. Using the Cancellation Graph to Guide Generalization

Physics 101 performs generalization by using its explanation of the specific solution (the *cancellation graph*) to guide the determination of the problem's unknown in the general case. This entails reconstructing the specific solution in its general form under the guidance of the

cancellation graph. In other words, the structure of the explanation of the specific problem's solution is generalized. The three-body collision example is used to illustrate the generalization process.

The system starts with the generalized unknown, $V^{\gamma_n}_{\gamma_s, \gamma_c}(\gamma_{arg})$.¹ It then performs the general versions of the specific formulae substitutions that produced the first of the primary obstacles. This can be seen, for the collision problem, in figure 4.1.²

While the general equation schemata are being applied, a global unification list is maintained, in the manner of the EGGS system [Mooney86]. This process determines how the terms in the new general formulae used must relate to ones already in the general calculation. For example, γ_{arg} in the generalized unknown is constrained to be t and γ_n is constrained to be 1, since the first step of figure 4.1 applies the second equation of figure 2.4 to the generalized unknown. Unifications that are needed to satisfy the preconditions of the equation schemata are also maintained.

$$\begin{aligned}
 & V_{\gamma_s, \gamma_c}(t) \\
 (1) \quad & = \int A_{\gamma_s, \gamma_c}(t) dt \\
 (2) \quad & = \int \frac{F_{net, \gamma_s, \gamma_c}(t)}{M_{\gamma_s}} dt \\
 (3) \quad & = \frac{1}{M_{\gamma_s}} \int F_{net, \gamma_s, \gamma_c}(t) dt \\
 (6) \quad & = \frac{1}{M_{\gamma_s}} \int (F_{ext, \gamma_s, \gamma_c}(t) + F_{int, \gamma_s, \gamma_c}(t)) dt \\
 (7) \quad & = \frac{1}{M_{\gamma_s}} \int (F_{ext, \gamma_s, \gamma_c}(t) + \sum_{\substack{j \in \text{ObjectsInWorld} \\ j \neq \gamma_s}} F_{j, \gamma_s, \gamma_c}(t)) dt
 \end{aligned}$$

Figure 4.1 Introduction of the Primary Obstacles

¹ The variables used in this chapter are defined in section 2.5.

² The figures that follow, except figure 4.5, are verbatim transcriptions of actual outputs of the implemented system. The numbers associated with each line refer to the calculation steps of figure 3.5.

Recall from section 3.4 that the inter-object forces are *additively* cancelled in the specific case. Hence, the next generalization step is to additively isolate each inter-object force. M_{γ_s} is introduced into the left-hand side of the general calculation in order to accomplish this isolation. Figure 4.2 presents this generalization step.

At this point the general versions of the primary obstacles are isolated for additive cancellation. To perform this cancellation, those terms that will cancel the intra-object forces must be introduced into the general calculation. The system determines that in the specific solution each inter-object force acting on ball 1 is cancelled by the equal-but-opposite inter-object force specified by Newton's third law.

In the general case, *all* of the other objects in a situation exert an inter-object force on object γ_s . *All* of these inter-object forces need to be cancelled. In the specific case, $M_2 \times V_2$ produced and isolated the additive canceller of $F_{2,1}$ while $M_3 \times V_3$ produced and isolated the additive canceller of $F_{3,1}$. So to cancel object γ_s 's inter-object forces, an $M_j \times V_j$ term must come from every other object in the situation. Figure 4.3 presents the introduction of the

$$\begin{aligned}
 & M_{\gamma_s} V_{\gamma_s, \gamma_c}(t) \\
 (1) \quad &= M_{\gamma_s} \int A_{\gamma_s, \gamma_c}(t) dt \\
 (2) \quad &= M_{\gamma_s} \int \frac{F_{net, \gamma_s, \gamma_c}(t)}{M_{\gamma_s}} dt \\
 (3) \quad &= \frac{M_{\gamma_s}}{M_{\gamma_s}} \int F_{net, \gamma_s, \gamma_c}(t) dt \\
 (4) \quad &= 1 \int F_{net, \gamma_s, \gamma_c}(t) dt \\
 (5) \quad &= \int F_{net, \gamma_s, \gamma_c}(t) dt \\
 (6) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + F_{int, \gamma_s, \gamma_c}(t)) dt \\
 (7) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + \sum_{\substack{j \in \text{ObjectsInWorld} \\ j \neq \gamma_s}} F_{j, \gamma_s, \gamma_c}(t)) dt
 \end{aligned}$$

Figure 4.2 Introduction of M_{γ_s} to Additively Isolate the Primary Obstacles

summation that produces the terms that cancel object γ_s 's inter-object forces. Notice how the goal of cancellation motivates generalizing the number of objects involved in this expression.

Once all the cancellers of the generalized primary obstacle are present, the primary obstacle itself can be cancelled. This is shown in figure 4.4, which is a continuation of figure 4.3 (the last line of figure 4.3 is repeated in figure 4.4).

$$\begin{aligned}
 & M_{\gamma_s} V_{\gamma_s, \gamma_c}(t) + \sum_{\substack{j \in \text{ObjectsInWorld} \\ j \neq \gamma_s}} M_j V_{j, \gamma_c}(t) \\
 (1) \quad &= M_{\gamma_s} \int A_{\gamma_s, \gamma_c}(t) dt + \sum_{j \neq \gamma_s} M_j \int A_{j, \gamma_c}(t) dt \\
 (2) \quad &= M_{\gamma_s} \int \frac{F_{net, \gamma_s, \gamma_c}(t)}{M_{\gamma_s}} dt + \sum_{j \neq \gamma_s} M_j \int \frac{F_{net, j, \gamma_c}(t)}{M_j} dt \\
 (3) \quad &= \frac{M_{\gamma_s}}{M_{\gamma_s}} \int F_{net, \gamma_s, \gamma_c}(t) dt + \sum_{j \neq \gamma_s} \frac{M_j}{M_j} \int F_{net, j, \gamma_c}(t) dt \\
 (4) \quad &= 1 \int F_{net, \gamma_s, \gamma_c}(t) dt + \sum_{j \neq \gamma_s} 1 \int F_{net, j, \gamma_c}(t) dt \\
 (5) \quad &= \int F_{net, \gamma_s, \gamma_c}(t) dt + \sum_{j \neq \gamma_s} \int F_{net, j, \gamma_c}(t) dt \\
 (6) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + F_{int, \gamma_s, \gamma_c}(t)) dt + \sum_{j \neq \gamma_s} \int (F_{ext, j, \gamma_c}(t) + F_{int, j, \gamma_c}(t)) dt \\
 (7) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + \sum_{j \neq \gamma_s} F_{j, \gamma_s, \gamma_c}(t)) dt + \sum_{j \neq \gamma_s} \int (F_{ext, j, \gamma_c}(t) + \sum_{k \neq j} F_{k, j, \gamma_c}(t)) dt
 \end{aligned}$$

Figure 4.3 Introduction of the Cancellers of the Primary Obstacles

$$\begin{aligned}
 (8) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + \sum_{j \neq \gamma_s} F_{j, \gamma_s, \gamma_c}(t)) dt + \sum_{j \neq \gamma_s} \int (F_{ext, j, \gamma_c}(t) + \sum_{k \neq j} F_{k, j, \gamma_c}(t)) dt \\
 (9) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + \sum_{j \neq \gamma_s} F_{j, \gamma_s, \gamma_c}(t) + \sum_{j \neq \gamma_s} [F_{ext, j, \gamma_c}(t) + \sum_{k \neq j} F_{k, j, \gamma_c}(t)]) dt \\
 (10) \quad &= \int (F_{ext, \gamma_s, \gamma_c}(t) + 0 \frac{kg \cdot m}{s^2} + \sum_{j \neq \gamma_s} [F_{ext, j, \gamma_c}(t) + \sum_{k \neq j} F_{k, j, \gamma_c}(t)]) dt
 \end{aligned}$$

Figure 4.4 Cancellation of the Primary Obstacles

Now that the primary obstacles are cancelled, the system checks to see if any secondary obstacles have been introduced. As can be seen in figure 4.4, the inter-object forces *not* involving object $?s$ still remain in the expression. These are secondary obstacles. Figure 4.5 graphically illustrates these remaining forces in a situation containing N objects. All of the forces acting on object $?s$ have been cancelled, while a force between objects j and k still appears whenever neither j nor k equal $?s$. This highlights an important aspect of generalizing to N . Introducing more entities may create new interactions that do not appear and, hence, are not addressed in the specific example. This issue is further elaborated in the discussion of the BAGGER system.

Physics 101 cannot eliminate the remaining inter-object forces if the specific example only involves a two-object collision. The system does not detect that the remaining forces all cancel one another, since in the two-object example there is no hint of how to deal with these secondary obstacles. A collision involving three or more must be analyzed by the system to properly motivate this cancellation. More details on the reasons for this are given in chapter 7, which presents additional examples. In the three-body collision problem, the system continues as shown in figure 4.6.

Once all possible obstacle cancellations of the cancellation graph have been produced, **Physics 101** produces the final result. The preconditions of each equation schemata are collected, the global unification list is used to determine the final form of each variable in these preconditions, and the final result is simplified. This process produces the restrictions that the

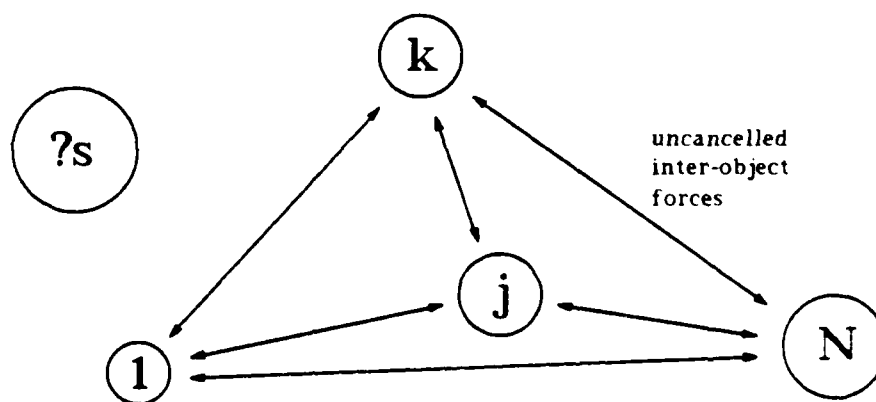


Figure 4.5 The Remaining Inter-Object Forces

$$(10) = \int (F_{ext, \gamma_s, \gamma_c}(t) + 0 \frac{kg \cdot m}{s^2} + \sum_{j \neq \gamma_s} [F_{ext, j, \gamma_c}(t) + 0 \frac{kg \cdot m}{s^2}]) dt$$

$$(11) = \int (F_{ext, \gamma_s, \gamma_c}(t) + \sum_{j \neq \gamma_s} F_{ext, j, \gamma_c}(t)) dt$$

Figure 4.6 Cancellation of the Secondary Obstacles

masses of the objects be constant over time (since each was factored out of a temporal integral - see figure 3.5), and that the objects cannot have zero mass (since their masses appear in the denominator of expressions). The final result is shown in figure 4.7. The new equation is recorded, along with its preconditions. In addition, the terms cancelled in the general calculation are recorded. Although not implemented in **Physics 101**, the eliminated terms could be used to help index the acquired formula. For example, when the inter-object forces are not specified, this equation schema could be suggested as possibly being appropriate.

In addition to not being restricted to situations containing three objects, the newly-acquired formula is not restricted to situations where the external forces are all zero. Instead,

Equation

$$\frac{d}{dt} \sum_{i \in ObjectsInWorld} M_i V_{i, \gamma_c}(t) = \sum_{i \in ObjectsInWorld} F_{ext, i, \gamma_c}(t)$$

Preconditions

$$\begin{aligned} & \text{IsaComponent}(\gamma_c) \wedge \\ & \forall i \in ObjectsInWorld \text{ NOT}(\text{ZeroValued}(M_i)) \wedge \\ & \forall i \in ObjectsInWorld \text{ IndependentOf}(M_i, t) \end{aligned}$$

Eliminated Terms

$$\forall i \forall j \neq i F_{i, j, \gamma_c}(t)$$

Figure 4.7 The Final Result

an appreciation of how the external forces effect momentum is obtained. This process also determines that there is no constraint that restricts this formula to the x -direction. It applies equally well to the y - and z -components of V . Hence, the acquired formula is a vector law. Notice that those physics variables whose values are used in the specific solution (e.g., the F_{ext}) remain in the general formula. The final equation is added to **Physics 101**'s collection of general formulae. (If **Physics 101** generalizes the two-body collision it would produce in an expression still containing those inter-object forces that do not involve object i . However, this formula would not be kept. See section 7.4 for further discussion.) The new formula says: *The rate of change of the total momentum of a collection of objects is completely determined by the sum of the external forces on those objects.* Other problems, which involve any number of bodies under the influence of external forces, can be solved by the system using this generalized result. For example, it can be used to solve the three-dimensional collision problem involving four objects, where there are external forces due to gravity, that is shown in figure 4.8.

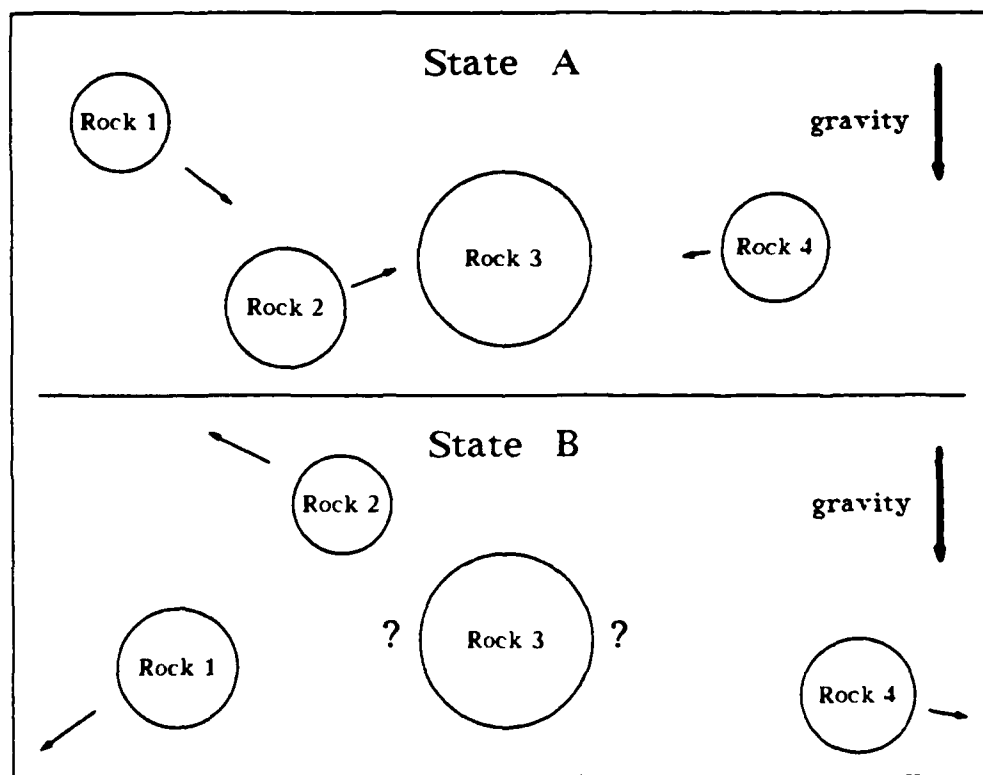


Figure 4.8 A Three-Dimensional Collision Problem in a Gravitational Field

Not all of the preconditions of the equation schemata in a calculation appear in the final result. Each equation schema providing support in a calculation may have associated with it propositions that are known to be true. *Known facts* are filtered out of the final result. For example, if $\sum_{\substack{j \in \text{ObjectsInWorld} \\ j \neq ?i}}$ appears in an equation schema, that schema's known facts include $\text{Member}(j, \text{ObjectsInWorld})$ and $j \neq ?i$. Preconditions of the equation schemata used in a calculation that match any of the known facts do not appear in the final collection of preconditions. For instance, a precondition of the equation $F_{j, ?i} = -F_{?i, j}$ is $j \neq ?i$. This precondition does not appear in figure 4.7 because the use of another schema leads to this precondition being a known fact.

4.2. The Result of Standard Explanation-Based Learning

Standard explanation-based learning generalization algorithms (e.g., [Mitchell86, Mooney86]) can be applied to the explanation structure underlying a calculation. The result obtained by doing this with the calculation of figure 3.5 is presented in figure 4.9.

A number of characteristics of the sample problem are generalized, many with the same result as in the **Physics 101** system. For example, the x -component of the velocities need not be used. The technique applies to any vector component. Also, because of the equation schemata used, the masses of the three objects must be non-zero and constant with respect to time. The cancellations of the inter-object forces (line 10 of figure 3.5) leads to the requirement that the general objects be distinct.³ The property of the specific example that each external force is zero is generalized. In the general case these forces need not *individually* be zero. What is needed is that they *sum* to zero. Line 15 of figure 3.5 produces this constraint because the integration rule used requires that the integrand be zero.

However, because the *structure* of the explanation is not generalized using standard generalization algorithms, the result obtained in this manner only applies to situations where there are *three* objects in the problem's world. A separate rule must be learned for situations containing four objects, five objects, etc. In addition, the acquired schema is not relevant when the external forces do not sum to zero. No appreciation of the effect of external forces on a

³ This constraint does not appear in the result of **Physics 101** because it is a *known fact*.

Equation

$$\begin{aligned} & M_{\gamma_{i_1}} V_{\gamma_{i_1}, \gamma_c} (?t_1) + M_{\gamma_{i_2}} V_{\gamma_{i_2}, \gamma_c} (?t_1) + M_{\gamma_{i_3}} V_{\gamma_{i_3}, \gamma_c} (?t_1) \\ &= M_{\gamma_{i_1}} V_{\gamma_{i_1}, \gamma_c} (?t_2) + M_{\gamma_{i_2}} V_{\gamma_{i_2}, \gamma_c} (?t_2) + M_{\gamma_{i_3}} V_{\gamma_{i_3}, \gamma_c} (?t_2) \end{aligned}$$

Preconditions

$$\begin{aligned} & \text{IsaComponent}(\gamma_c) \wedge \\ & ?t_1 \neq ?t_2 \wedge \\ & \text{NOT}(\text{ZeroValued}(M_{\gamma_{i_1}})) \wedge \\ & \text{NOT}(\text{ZeroValued}(M_{\gamma_{i_2}})) \wedge \\ & \text{NOT}(\text{ZeroValued}(M_{\gamma_{i_3}})) \wedge \\ & \text{IndependentOf}(M_{\gamma_{i_1}}, t) \wedge \\ & \text{IndependentOf}(M_{\gamma_{i_2}}, t) \wedge \\ & \text{IndependentOf}(M_{\gamma_{i_3}}, t) \wedge \\ & ?i_1 \neq ?i_2 \wedge \\ & ?i_1 \neq ?i_3 \wedge \\ & ?i_2 \neq ?i_3 \wedge \\ & \text{Permutation}(\{?i_1 ?i_2 ?i_3\}, \text{ObjectsInWorld}) \wedge \\ & \text{ZeroExpression}(\text{ValueOf}(F_{ext, ?i_1, \gamma_c}) + \text{ValueOf}(F_{ext, ?i_2, \gamma_c}) \\ & \quad + \text{ValueOf}(F_{ext, ?i_3, \gamma_c})) \end{aligned}$$

Figure 4.9 The Result of Standard Explanation-Based Learning

system's momentum is obtained. By recognizing and analyzing obstacle cancellations, then reconstructing the explanation in the general case, **Physics 101** overcomes these shortcomings.

4.3. Why the Specific and Generalized Calculations can Differ

As illustrated by the momentum example, applying standard explanation-based generalization algorithms to the specific example is not sufficient because the structure of the calculation is not generalized. In **Physics 101** generalization occurs by reconstructing the specific calculation in the general case, thereby generalizing the structure of the specific example. There are three reasons why the underlying structure of the generalized calculation can differ from that of the specific calculation.

- (1) One cause of altering structure is the fact that some equation schemata are marked "problem-specific." Schemata that replace variables (e.g., F_{ext}) with their values in a specific problem are the simplest type of problem-specific schemata. More complicated examples are also possible. For example, the formula $F_{net} = M A$ hold in all physics problems, while $F = M g$ is only applicable to certain problems. Problem-specific equations are not used when generalizing a calculation (they may be used, though, when building special cases). The obstacle graph helps determine how the calculation can be reconstructed without using equation schemata of this type. How this is done is explained later, and a sample problem where this occurs appears in section 7.1.
- (2) Another manner of altering calculation structure is mentioned in section 3.4, where the algorithm for constructing cancellation graphs is first outlined. Any variables appearing in the left-hand side of a specific calculation, but not appearing in the calculation's cancellation graph, do not appear in the generalized calculation. The generalization algorithm never considers these variables nor their descendants.
- (3) The third cause is the primary reason for the proper generalization of the momentum example. As described earlier, in specific problems "expanded" versions of equations are used, while in the general calculation the unexpanded forms are used. For example, the general equation

$$A(x) = \prod_{i=1}^N B(x, y_i) \quad \text{expands to} \quad A(x) = B(x, y_1) B(x, y_2) B(x, y_3) B(x, y_4)$$

in a problem where $N = 4$. This is what facilitates the generalization of number. Notice, though, that simply replacing an expanded equation by a general equation containing a \sum

or a \prod does not produce the proper generalization. As in the momentum example, the occurrence of a summation deep within the calculation leads to the summation of some previously ungrouped terms (i.e., the $M_i V_i$'s). Focussing on obstacles and their elimination motivates this global regrouping.

If specific equations are replaced by their general versions in the momentum example, and constants replaced by constrained (predicate calculus) variables as in standard explanation-based learning algorithms, then equation 4.1 would arise during the generalization of the calculation in figure 3.5.⁴ The specific example only involves the cancellation of *two* inter-object forces acting on each ball. Hence, two inter-object forces are cancelled from each summation. In the next step of the standard generalization process, the integrand in the right-hand side of equation 4.1 would be constrained to be zero, to insure that the result of integration would be the required constant.

The final result would be correct. Whenever this constraint and other preconditions are satisfied, the sum of the three $M V$ terms is constant. However, this result is overly restrictive — it does not represent the physical concept of conservation of momentum. These extra summations are not constrained by Newton's laws to always be zero. Only in particular circumstances will they be so. The correct concept requires the momenta of *all*

$$\begin{aligned}
 & M_{i_1} V_{i_1, 2c}(t) + M_{i_2} V_{i_2, 2c}(t) + M_{i_3} V_{i_3, 2c}(t) \\
 &= \int (F_{ext, i_1, 2c}(t) + \sum_{j=i_1, i_2, i_3} F_{j, i_1, 2c}(t) + \\
 &\quad F_{ext, i_2, 2c}(t) + \sum_{j=i_2, i_1, i_3} F_{j, i_2, 2c}(t) + \\
 &\quad F_{ext, i_3, 2c}(t) + \sum_{j=i_3, i_1, i_2} F_{j, i_3, 2c}(t)) dt
 \end{aligned} \tag{4.1}$$

⁴ Physics 101 has not be modified to operate in this manner. The discussion about equation 4.1 is theoretical.

the objects in a situation be added. In this case, the inter-object forces cancel each other, regardless of their value.

In summary, to produce the desired result, the left-hand side of the acquired equation must contain the sum of the $M V$ terms for each object in the situation. There must be proper motivation for this regrouping of an expression a large distance away from the expression where the inter-object forces cancel. Simply replacing equations in the specific calculation with their general versions, and then applying standard explanation-based constraint propagation will not result in the correct regrouping of the left-hand side. The necessary regrouping will also not occur if the problem solver uses the general formulae, rather than the expanded versions, when constructing the original justification of the teacher's solution step. Focussing on the cancellation of obstacles *does* provide the necessary guidance, by determining the role of these $M V$ terms in the specific example and then extending the role to the general case.

4.4. Learning Special-Case Schemata

One would expect that acquired schemata should be as general as possible so that they might each cover the broadest class of future problems. Indeed, explanation-based learning research has been primarily aimed at the acquisition of such maximally general schemata. However, an intermediate level of generalization is often appropriate. The class of intermediate generality schemata improves the performance of a system's problem solver by supplying "appropriately general" schemata instead of forcing the system to rely on its maximally general schemata. Storing both the maximally general schemata and the intermediate level schemata results in much improved efficiency with no loss of generality. This section describes how **Physics 101** produces intermediate level schemata, which are called *special cases*.

New schemata are learned by **Physics 101** as described in the sections above. As well as storing the new schema in its general form, the system constructs and stores special cases. These special cases are the result of composing the new, general equation schema with a problem-solving schema. A successful composition results in a specialization which is guaranteed to work using the composed problem-solving technique. This frees the problem solver from performing the planning that would otherwise be required to elaborate the general schema to fit

the current problem-solving episode. The system can, of course, always resort to its collection of maximally general schema when no special case is appropriate.

As illustrated by the result in figure 4.7, the explanation-based generalization of a sample collision problem leads to a physics formula that describes how external forces change a system's momentum. This general schema is broadly-applicable, but ascertaining that it will lead to the solution of a given problem requires a good deal of work. The external forces must be summed, then integrated, and the constant of integration must be determined by using some initial conditions. All of this work can be pre-packaged into a special-case schema, if assumptions about the external forces are made.

A major issue in explanation-based learning concerns the *operationality/generalizability* trade-off [DeJong86, Keller87b, Mitchell86, Segre87b, Shavlik87e]. A schema whose relevance is easy to determine may only be useful in an overly-narrow range of problems. Conversely, a broadly-applicable schema may require extensive work before a problem solver can recognize its appropriateness. Most approaches to selecting the proper level of generality involve pruning easily-reconstructable portions of the explanation structure. In **Physics 101**, operational rules are produced by constraining a general schema in such a way that its relevance is easily checked.

The intermediate level schemata generated by **Physics 101** are similar in scope of applicability to those that human experts appear to possess. For example, the conservation of momentum problem results in a special-case schema characterized by the absence of external forces and the specification of a *before* and *after* situation. These features are those cited by experts as the relevant cues for the principle of conservation of momentum (see table 12 of [Chi81]).⁵

Although the motivation for this intermediate level of generalization is computational, the use of this level helps to reconcile the approach with a variety of psychological evidence showing that problem solvers use highly specific schemata [Chase73, Hinsley77, Schoenfeld82, Sweller85]. Much of expertise consists of rapidly choosing a tightly-constrained schema appropriate to the current problem. However, the difference between the knowledge of an expert and a novice cannot be explained on the basis of number of schemata alone. The scope

⁵ It should be noted that it was not the explicit intent to model this psychological data. Rather, computational efficiency considerations led to a system that produced results matching this empirical data.

and organization of these schemata have been shown in psychological experiments to be qualitatively different [Chi81, Larkin80, Schoenfeld82]. In representing a problem, novices make great use of the specific objects mentioned in the problem statement, while experts first categorize according to the techniques appropriate for solving the problem.

Extending Physics 101's Learning Model

Figure 4.10 contains an overview of the **Physics 101** system architecture, extended to produce special cases. As in the basic model, a known problem-solving schema is initially used to understand a solution to a specific problem. The explanation-based analysis of the solution may lead to the construction of a new broadly-applicable schema. The generalization process often produces a new schema that, in its fullest form, is not usable by the originally applied problem-solving schema. For example, the acquired momentum law describes how the external forces affect a physical system's momentum. It is *not* a conservation law, although the original calculation involved the problem-solving schema for conserved quantities. The new phase added to the learning model occurs next. Constraining the general result so that the originally-used problem-solving schema does apply produces a special case. In the special-case schema, the

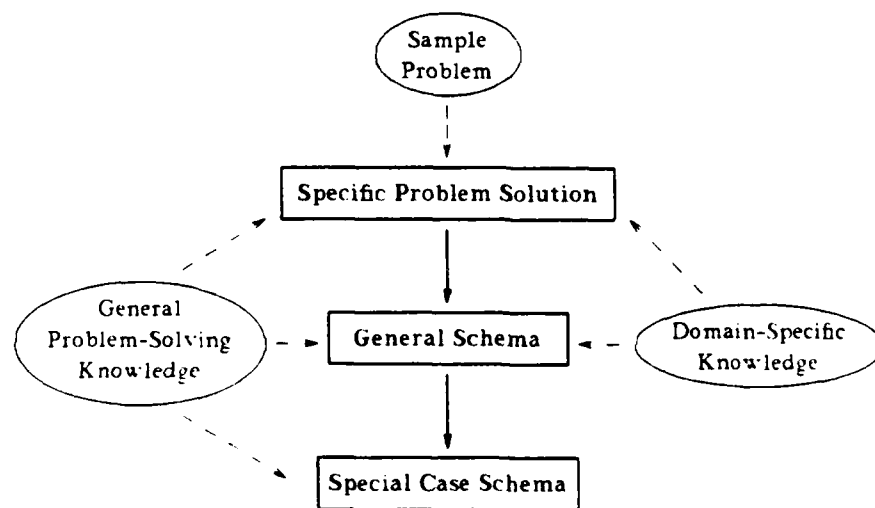


Figure 4.10 Overview of the Extended Learning Model

constrained schema, its constraints, and the original problem-solving schema are packaged together to produce a specialized equation schema.⁶

In the approach taken in the **Physics 101** system, automatically acquiring schemata of the intermediate level of generality requires that the system's schemata be organized into two classes:

- (1) Schemata that represent general problem-solving knowledge, which apply across many application domains (e.g., a schema for utilizing a conserved quantity to solve a problem). In **Physics 101**, these are the *problem-solving schemata*.
- (2) Schemata that represent declarative knowledge of the domain of application (e.g., Newton's laws). In **Physics 101**, these are the *equation schemata*.

Humans with mature problem-solving backgrounds possess the first type of schema and are told schemata of the second type when introduced to a new domain. Through study they acquire a large collection of schemata that combine aspects of both types, thereby increasing their problem-solving performance in the domain. Combining general problem-solving techniques with domain specific knowledge produces schemata that, when applied, lead to the rapid solution of new problems. It is this performance that is being modelled in the **Physics 101** system.

Figure 4.11 shows the relation between a general schema and its special cases. (Although not shown in the figure, there could be special cases of the special cases.) The properties that distinguish the special cases from their general cases are recorded.⁷ If a special case is not applicable, the general concept is then accessed. Besides being constructed when the general case is acquired, a new special case could be created whenever the general case is used to solve a later problem.

⁶ Only one special-case is constructed. At the end of this section, methods for producing multiple special cases from one problem are discussed.

⁷ The method of using these special-case cues to select the appropriate schema is not addressed in this research. Possible indexing techniques include approaches based on *discrimination nets* [Feigenbaum63, Kolodner84, Schank82] and approaches based on *spreading activation* [Anderson83, Quillian68]. The next chapter discusses how **Physics 101** uses special cases in problem solving.

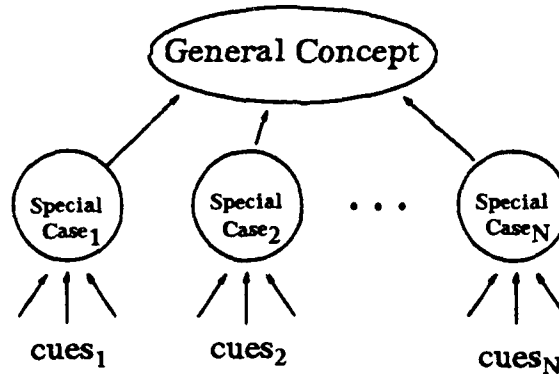


Figure 4.11 Inter-Schema Organization

To construct a special case, **Physics 101** first satisfies the preconditions of the problem-solving schema used in the specific problem. The newly-acquired equation schema is the preferred equation schema to be used during this process, and any of the characteristics of the specific problem are acceptable. The resulting proof true is generalized using a standard explanation-based learning algorithm. The leafs of the generalized proof tree constitute the necessary requirements for using the problem-solving schema with the new equation schema. Those general preconditions not known to be always true, nor known to be easily derivable, are combined with the preconditions of the general schema to produce the special case's preconditions.

There are a number of ways that special cases can be constructed from a general schema. One approach would be to try combining *all* known problem-solving schema with the general result, storing all successful combinations. However, a focus in this research is to avoid such exhaustive algorithms. Another approach would be to take advantage of the hierarchy of problem-solving schemata and only consider schemata within some arbitrary distance of the schema actually used. Since no theoretically acceptable way to limit this distance has been developed, attention only focusses on the schema used in the specific problem-solving episode.

The properties of the sample problem that allowed the problem-solving schema to apply are used to characterize the special-case schema. This can be more restrictive than the most general special case. For example, in the momentum problem, all that is needed is that the external forces *sum* to zero, and not that there be *no* external forces. However, in the sample problem, the integration of the external forces is supported by the fact that the external forces

are individually zero, and this property is used to build the special case. This relates to the idea of sample problems being representative of future problems and to the goal that the preconditions of special cases be easily evaluated (and also matches psychological data). One extension of the approach taken would be to store both the most general special case (i.e., the special case that results from constraining the general schema just enough so that the previously-used problem-solving schema applies) and the special case that results from seeing how the preconditions for the most general special case are satisfied in the specific problem. Given an indexing scheme that has access time logarithmic in the number of schemata, the storage of extra schemata should not substantially hinder future problem solving.

In addition to improving a problem solver's efficiency, special cases also indicate good assumptions to make. For instance, if the values of the external forces are not known, assume they are zero, as this will allow easy solution to the problem. Physics problems often require one to assume things like "there is no friction", "the string is massless", "the gravity of the moon can be ignored", etc. Problem descriptions given in textbooks contain cues such as these, and students must learn how to take advantage of them. Facts in the initial problem statement suggest possible problem-solving strategies, while any additional requirements of the special case situations indicate good assumptions to make (provided they do not contradict anything else that is known).

An Example

This section presents an example of the construction of special-case schemata, again using the domain of classical physics. Further discussion of special cases appears in section 7.1, where a special case that results from an energy conservation example is presented.

The schema that makes use of conserved quantities during problem solving is shown in figure 4.12. It says that one way to solve for an unknown is to find an expression, containing the unknown, that is constant with respect to some variable, instantiate this expression for two different values of the variable, create an equation from the two instantiated expressions, and then solve the equation for the unknown. If the values of all but one variable at these two points are known, simple algebra can be used to easily find the unknown. This schema is used in understanding the teachers' solution shown in figure 3.3.

Preconditions

CurrentUnknown(*?unknown*) \wedge
ConstantWithRespectTo(*?expression*, *?x*) \wedge
SpecificPointOf(*?x₁*, *?x*) \wedge
SpecificPointOf(*?x₂*, *?x*) \wedge
?x₁ \neq *?x₂* \wedge
?leftHandSide = InstantiatedAt(*?expression*, *?x₁*) \wedge
?rightHandSide = InstantiatedAt(*?expression*, *?x₂*) \wedge
?equation = CreateEquation(*?leftHandSide*, *?rightHandSide*) \wedge
ContainedIn(*?unknown*, *?equation*)

Schema Body

Solve(*?equation*, *?unknown*)

Figure 4.12 Conserved Quantity Schema

Notice that the result in figure 4.7 is *not* a conservation law. It describes how the momentum of a system evolves over time. Although this new formula applies to a large class of problems, recognizing its applicability is not easy. The external forces on the system must be summed and a possibly complicated differential equation needs to be solved. Applying this law requires more than using simple algebra to find the value of the unknown.

A portion of the proof that the originally used problem-solving schema (figure 4.12) can be used with the new general formula appears in figure 4.13.⁸ In order for the conserved quantity schema to be applicable to this new formula, it must be the case that momentum be constant with respect to time. This means that the derivative of momentum be zero, which leads to the requirement that the external forces sum to zero. This requirement is satisfied in the specific solution because each external force is individually zero, and this property is used to characterize the special case. When this occurs, the momentum of a system can be equated at any two distinct states. The special case schema for momentum conservation is contained in figure 4.13. Besides adding extra preconditions to those of the general schema, two more terms

⁸ Arrows run from the antecedents of an inference rule to its consequents.

$$\begin{array}{c}
 \text{ConstantWithRespectTo}(\sum_i M_i V_{i, \gamma_c(t)}, t) \\
 \uparrow \\
 \frac{d}{dt} \sum_i M_i V_{i, \gamma_c(t)} = 0 \frac{\text{kg m}}{\text{s}^2} \\
 \uparrow \\
 \frac{d}{dt} \sum_i M_i V_{i, \gamma_c(t)} = \sum_i F_{i, \text{ext}, \gamma_c(t)} \qquad \sum_i F_{i, \text{ext}, \gamma_c(t)} = 0 \frac{\text{kg m}}{\text{s}^2} \\
 \uparrow \\
 \forall i \ F_{i, \text{ext}, \gamma_c(t)} = 0 \frac{\text{kg m}}{\text{s}^2}
 \end{array}$$

Figure 4.13 Satisfying a Precondition of the Conservation Schema

Equation

$$\sum_{i \in \text{ObjectsInWorld}} M_i V_{i, \gamma_c} (?t_1) = \sum_{i \in \text{ObjectsInWorld}} M_i V_{i, \gamma_c} (?t_2)$$

Preconditions

$$\begin{array}{l}
 \text{IsaComponent}(\gamma_c) \wedge \\
 \forall i \in \text{ObjectsInWorld} \ \text{NOT}(\text{ZeroValued}(M_i)) \wedge \\
 \forall i \in \text{ObjectsInWorld} \ \text{IndependentOf}(M_i, t) \wedge \\
 \text{SpecificPointOf}(?t_1, t) \wedge \\
 \text{SpecificPointOf}(?t_2, t) \wedge \\
 ?t_1 \neq ?t_2
 \end{array}$$

Eliminated Terms

$$\forall i \ \forall j \neq i \ F_{i, j, \gamma_c}(t), \quad ?t_1, \quad ?t_2$$

Special Case Conditions

$$\forall i \in \text{ObjectsInWorld} \ F_{\text{ext}, i, \gamma_c}(t) = 0$$

Figure 4.14 The Special-Case Momentum Law

are eliminated in producing the special case. Since this is a conservation schema, the time at which each state occurs need not be provided in a problem for this schema to apply.

4.5. Performance Analysis

Physics 101's problem solving improvement after learning is analyzed in this section. Performance on several collision problems, differing as to the number of physical objects involved, is measured before and after learning the concept of momentum conservation. The goal in each case is to determine the velocity of one of the objects after the collision. In all of the problems, there are no external forces, and sufficient, randomly-generated mass and velocity values are provided to make each problem soluble. Nothing is stated about the inter-object forces. As mentioned in the previous chapter, and further detailed in the next, the standard system cannot solve collision problems before learning due to the incompleteness of its problem solver. However, to gather the data presented below, the problem solver was slightly extended. This was done by allowing as many "blind" substitutions as necessary. Ordinarily a search path terminates before two of these unmotivated substitutions would occur consecutively.

The time to solve problems as a function of the number of objects in the situation is graphed in figure 4.15. The solid line represents the system's performance without learning, while the dashed line represents its performance after learning. Without benefit of learning, the system uses only its mathematical knowledge and the equations of figure 2.4. The special case law for momentum conservation is used to solve the problem after learning. (How the system selects this law is described in the next chapter.) All points result from averaging five measurements, and the choice of the number of objects is made randomly in order to reduce the effect of inter-sample dependencies. It takes about 280 seconds to learn the general momentum evolution law and its special-case concept of momentum conservation from the teacher's solution to the three-body collision problem.

The results indicate that, without learning, the time taken by the altered problem solver to solve new problems grows *exponentially* with the size of the problem. Conversely, empirically the time grows *linearly* with the benefit of learning. (For a three-body problem, learning speeds up solving the problem by about a factor of ten.) This graph demonstrates one of the main advantages of explanation-based learning. Even when a problem solver has sufficient knowledge to solve a problem, an exponential amount of time may be needed, making solution infeasible.

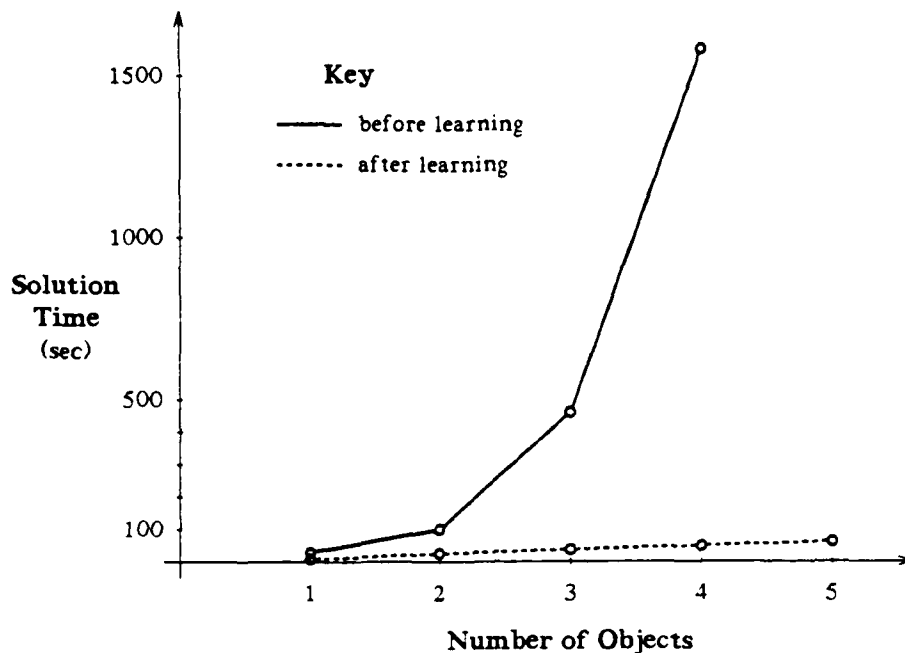


Figure 4.15 Performance on N-Body Collision Problems

The increase of efficiency provided by grouping inter-related schemata brings previously intractable classes of problems within the capabilities of the problem solver.

This graph demonstrates the value of a system that can generalize number. From one sample solution, a new concept is acquired that enables the system to rapidly solve collisions involving any number of objects. If **Physics 101** did not possess this capability, that is, if it used standard explanation-based learning techniques, it would still gain the speed-up of a factor of 10 on three-body collisions. However, it would require another sample solution when it experienced collisions involving different numbers of objects. Also, if the system acquired separate rules for each number of objects in a collision, a significant amount of problem-solving time could be wasted determining which, if any, is applicable to a new problem. Further empirical analysis of the the advantages of generalizing the structure of explanations is provided in chapter 11, which analyzes the performance of the **BAGGER** system.

4.6. Other Approaches to Learning in Mathematical Domains

There is a substantial literature describing computer models of learning in mathematical domains. In this section, these systems are described and compared to the approach taken in **Physics 101**.

BACON [Langley81, Langley87], **ABACUS** [Falkenhainer86], and **COPER** [Kokar86] empirically learn equations by analyzing multiple data measurements and discovering regularities. **AM** [Lenat76] discovers interesting concepts in mathematics, using heuristics to guide its searching. The **LEX** [Mitchell83b] and **Meta-LEX** [Keller87a, Keller87c] systems learn heuristics under which an integration operator is useful. *Versions spaces* [Mitchell78] are used in **LEX** to delimit plausible versions of the heuristic. This is done by considering positive and negative examples of application of the operator. **Meta-LEX** works by generating simplifications of its inference rules, then testing their effectiveness in improving the performance of problem solving. **DARWIN** [Araya84] uses experimentation, in place of a domain model, to determine when a special-case physics solution technique is applicable. It generates and tests variations on a problem description in order to construct a description of the class of problems soluble by the special-case technique used to solve the original problem. **PET** [Porter86] uses *experimental goal regression* to learn mathematics. This technique also achieves its power through directed experimentation.

Unlike **Physics 101**, these systems are inductive [Angluin83, Michalski83]. They learn new concepts that are consistent with the examples presented to or generated by them, by analyzing the similarities and differences among the examples. Confidence in the accuracy of the acquired results grows as more examples are seen. With only a few examples to analyze, irrelevant characteristics may appear irrelevant (because, for example, they appear in all of the positive examples seen) or necessary characteristics may appear unnecessary (because, for instance, they do not appear in any negative examples). **Physics 101**, being explanation-based, produces intertwined descriptions of the role of each aspect of a problem. These explanation structures allow the system to produce, from a small number of examples, new concepts that are as certain as the underlying concepts from which the explanation is constructed. The explanation-based approach allows **Physics 101** to incorporate into new concepts descriptions of the effects of problem characteristics not seen in the example from which it learns. For example,

although it experiences an example involving momentum conservation, it learns the more general equation describing how external forces affect momentum.

There have been a number of other explanation-based approaches to learning in mathematical domains. LEX2 [Mitchell83a] is an explanation-based version of LEX. LP [Silver86] analyzes worked mathematics problems and learns information that constrains, but does not eliminate, search through its operator space. ALEX [Neves85] learns to solve simple algebraic equations by examining solved problems. LA [O'Rourke87a] learns new schemata for use in natural deduction proofs. Finally, Bennett's system [Bennett87] learns approximations that transform intractable problems into soluble ones. However, none of these systems generalize the structure of their explanations.

Using analogy to learn new concepts in mathematical domains is another heavily investigated approach. Forbus and Gentner [Forbus83] describe an approach to learning *qualitative* physics using analogical reasoning. Phineas [Falkenhainer87] implements some of the ideas in their proposal. Aspects of geometry and algebra are analogically learned in the PUPS [Anderson87] system. The NLAG system [Griener88] learns about hydraulics using its knowledge of electricity.

In analogical learning, new knowledge is acquired by mapping old knowledge from a well-understood domain to a novel situation. Learning from analogies can involve aspects of the similarity/difference-based and explanation-based approaches to learning. Similarity between the current problem and previous problems helps in the selection of candidate analogs and in the construction of mappings between the two situations. Successful analogies may lead to the construction of abstractions of the two situations [Gick83, Winston82], which can be performed in a similarity or explanation-based fashion. One way analogy differs from the approach taken in explanation-based systems is that attention is not focussed on combining pieces of knowledge into larger knowledge chunks (schemata). Also, using analogy, no learning takes place the first time a problem is solved. Instead, an analogous problem must be encountered before the solution to the first problem contributes to the acquisition of knowledge. Finally, none of these systems address generalizing number.

4.7. Summary

This chapter presents the techniques by which **Physics 101** learns new general concepts by analyzing the solutions to specific problems. The generalization process is guided by the specific solution's *cancellation graph*. This data structure determines how the calculation is reconstructed, in a more general form. The general versions of the obstacles cancelled in the specific solution are also cancelled in the general calculation. Reconstruction of the calculation produces a generalization of the structure of the explanation of the original problem's solution. While this reconstruction takes place, constants in the example are converted to constrained variables using a standard explanation-based algorithm. No problem-solving search is performed during the construction of the general calculation — construction deterministically follows from the specific calculation. Hence, the process is relatively efficient. Special cases of the generalized calculation, which can be more efficiently used during problem solving, are also constructed.

Physics 101 offers a different perspective on the process of explanation-based generalization. Rather than directly using the explanation of the specific problem's solution as is done in more standard algorithms, the explanation of a specific calculation closely guides the construction of a general version of the calculation, from which a new general concept may be extracted. The new calculation is often substantially more general, in terms of its structure as well as its variables, than the specific calculation.

There are three reasons why the structure of a generalized calculation can differ from that of the specific calculation from which it is constructed. First, certain equations — those involving indefinite summations or products — are used in different forms in the two cases. Second, some equation schemata are problem-specific and cannot be used in the general calculation. Third, portions of the calculation that do not play any role in the elimination of obstacles do not appear in the general calculation.

From the sample collision problem used to illustrate generalization, a new schema is produced which describes how external forces affect the momentum of any collection of objects. Information about the number of entities in a situation, localized in a single physics formula, lead to a global restructuring of a specific solution's explanation. The effect of external forces is determined, even though the solution of the specific example took advantage of the fact that the external forces sum to zero. Also, the resulting equation applies to situations containing any

number of physical objects, while the specific example only contained three balls. Standard explanation-based learning algorithms do not produce these generalizations. If the structure of the specific solution's explanation were not generalized, the resulting new schema would only apply to situations containing three objects and where the external forces summed to zero. A separate rule would have to be learned each time a situation contained a different number of objects.

The performance increase achieved by learning this new schema is empirically measured. Before learning, the time it takes the problem solver (in a modified state) to solve new problems empirically grows *exponentially* with the size of the problem. The time grows *linearly* with the benefit of learning. On three-body problems similar to the original problem, there is a speed-up of about ten after learning.

Special case construction is a promising approach to the need for an intelligent system to acquire schemata at an appropriate level of generality. The schemata that result incorporate the constraints of a problem-solving schemata into a general schema. Additional features that a situation must possess if the special case is to apply are recorded, thereby reducing the amount of work needed to elaborate the schema in future problem-solving episodes.

A common induction scheme is to posit that learners compare particular instances of a concept (such as specific problems of a problem type) and abstract out those aspects that are common to both problems, e.g., [Anderson83, Michalski83, Mitchell78, Posner68]. The fact that problem solvers use highly specific schemata supports such a view, since these schemata would arise whenever two problems from an intermediate level problem type are compared. Although similarity-based generalization is an important means of learning, especially for human novices [Gentner88, Ross84, Ross88], the research in **Physics 101** shows that many of these highly specific schemata can arise from an explanation-based approach. Even some strong proponents of example comparison learning have begun to incorporate some explanation-based ideas in order to account for how much is learned from one example [Anderson88].

Because explanation-based learning requires extensive domain knowledge, it clearly is not appropriate for modelling all learning in a new domain. Nevertheless, it may be useful even in early learning if the domain relies heavily upon a domain for which the novice does have substantial knowledge. Because mathematics underlies many other domains, a novice with some

mathematical sophistication may be able to make use of explanation-based techniques without extensive knowledge of the new domain.

The next chapter (chapter 5) describes how the system's uses the schemata it learns in later problem solving. The chapter after that provides details of the algorithms and data structures used in **Physics 101**, while chapter 7 analyzes the generalization of several additional examples.

Chapter 5

Physics 101's Problem Solver

Enhancing problem-solving performance is one of the most important reasons for learning [Simon83]. This chapter discusses **Physics 101's** problem solver, including how acquired equation schemata are used in future problem solving. The problem solver serves two basic purposes in the system. One, it attempts to solve new problems, and, two, when it cannot solve a new problem, it builds the explanations that justify the steps in the solution provided by the system's user.

Physics 101 contains a schema-based problem solver which searches through the space of mathematical expressions. The problem solver is provided a goal description — a mathematical expression called the *unknown* and a description of the property of this unknown whose value is desired. For example, the unknown may be a single variable whose value is sought or it may be an algebraic expression where the goal is to determine the expression's time-dependence. The goal description can either come from the user, requesting information about some situation, or it can come from the system's understander module, in its efforts to justify a user-provided solution step. Following receipt of the goal description, the problem solver first chooses an initial equation and then successively applies legal transformations to the current right-hand

side of the equation, substituting sub-expressions for other sub-expressions, until an expression satisfying the goal description is reached. The overall process is sketched in figure 5.1. Notice that more than one substitution is allowed during the transformation from one equation to the next. A description of the problem solver is provided in this section. Further information is provided in section 6.3, which provides algorithmic details.

Figure 5.2 contains an overview of the operation of the problem solver after it is given a description of the current problem. (See chapter 2 for the definition of the task of problem solving.) There are three main stages involved in solving a problem. First, some equation schema must be chosen that appears to bear relevance toward the problem. This equation is algebraically manipulated so that all the *obstacles* appear on the right-hand side of the equation. Next, this right-hand side expression is transformed until all of the obstacles are eliminated. If the problem solver cannot do this, it attempts to choose an alternative starting equation. When this can no longer be done, a solution from the system's teacher is requested and analyzed, which may lead to new equation schemata being added to the database of schemata. If an acceptable equation is produced, the answer to the problem is derived from it in the last stage of problem solving.

A simple example illustrates these phases. Assume the problem is to determine the value of variable x , given the values of variables a , b , and c . Also assume the system possesses

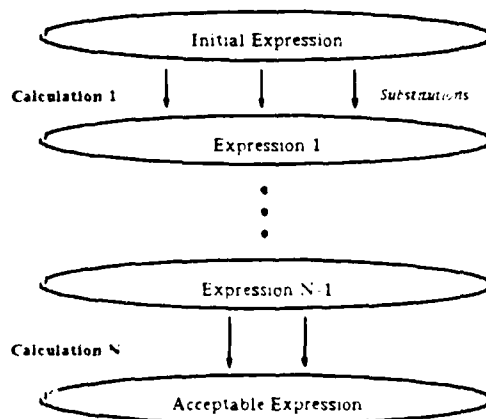


Figure 5.1 The Structure of a Calculation Sequence

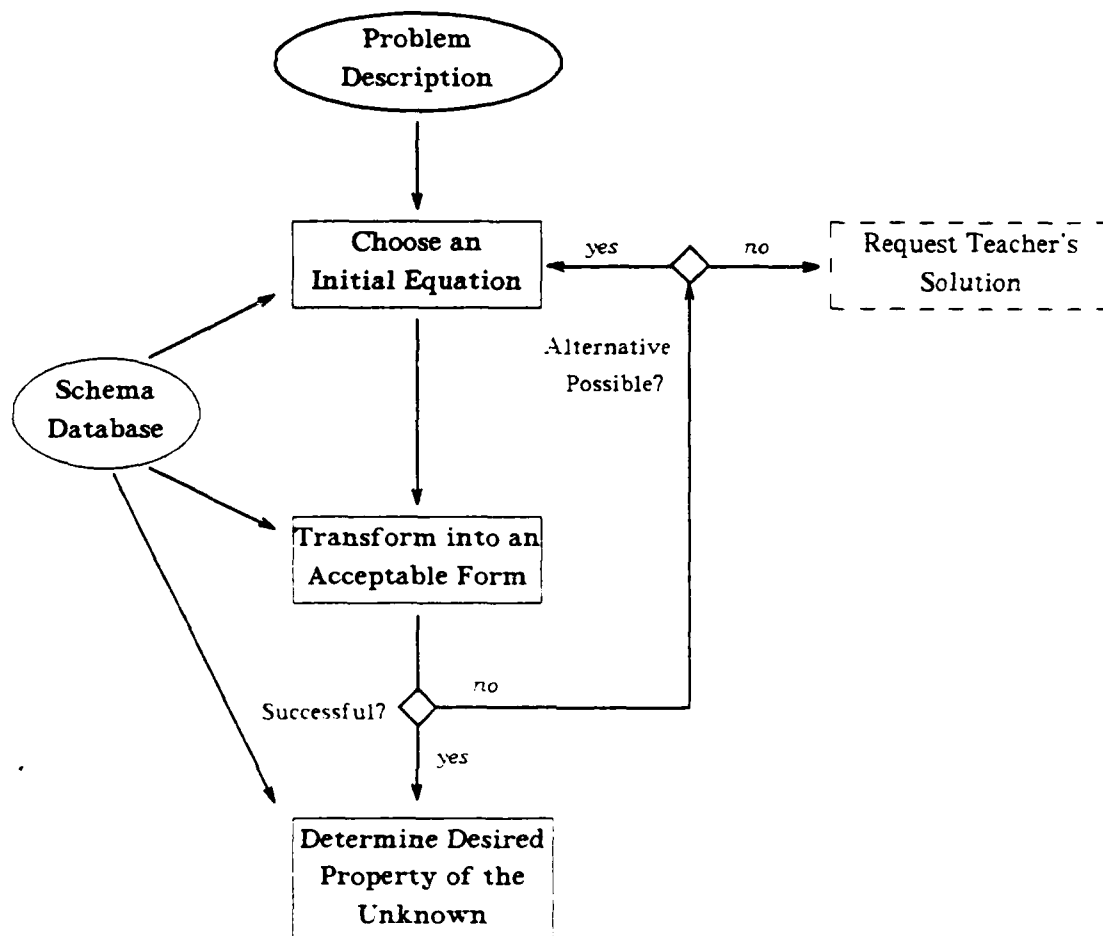


Figure 5.2 Overview of the Problem Solver

equation schemata containing the following equations.

$$y = a x + b \quad \text{and} \quad y = 10 c$$

Initially the system will choose the equation on the left and reverse it so that the obstacle y is on the right. Next, it will transform this unacceptable expression by using the equation $y = 10c$, producing $a x + b = 10 c$, an acceptable equation. Finally, the value of x is extracted.

Later sections of this chapter describe the methods by which **Physics 101** chooses an equation with which to work and how it searches the space of mathematical expressions in the

hopes of eliminating the obstacles. First, there is a brief description of schema-based problem solving.

5.1. Schema-Based Problem Solving

A schema-based problem solver attacks problems by combining known problem-solving schemata in order to achieve a desired goal. Schemata of this type specify the goal they achieve, the preconditions that must be true for the schema to be applicable, and either the intermediate sub-goals that must be met or the actual solution steps. This means that schemata can either decompose a problem or actually solve it. If no known schemata apply, the system is not able to solve the problem.

To illustrate these ideas, consider someone with the goal of buying a new suit. The preconditions for the *clothes-buying* schema would include such things as having enough money to buy a suit, time to go to a clothing store during its business hours, and a means of transportation. The plan would include traveling to the store, choosing a suit, paying for it, and returning. Other schemata might contain the details of how these steps could be performed.

A major claim of schema-based problem solvers is that intelligence consists of a large collection of general schemata together with the ability to recognize when to apply each, rather than the ability to construct novel, creative solutions to a problem. This view entails very little searching. Instead, power comes from the number and generality of a system's schemata. Having the right knowledge chunks is essential to effective problem-solving.

A major goal of schema-based problem solvers is to minimize the combinatorially-explosive nature of search-based problem solving. Schemata package together a collection of inter-related problem-solving operators in order to guide the search through the problem space. One schema can move the solver through several states in the problem space without intervening search.¹ This is illustrated in figure 5.3 where a portion of a search tree is drawn. A single schema may contain the composition of the operators indicated by bold lines, allowing directed traversal through this subtree.

¹ If none of the encapsulated operators performs an action on the external world, the problem solver can jump directly from the current state to the one that results from applying the sequence of operators.

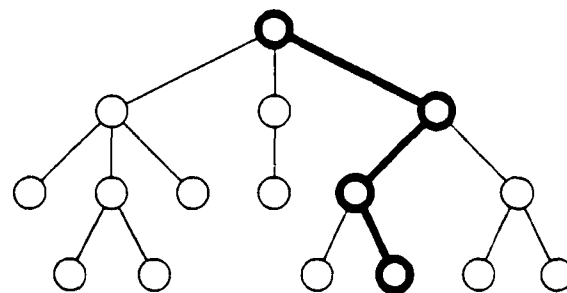


Figure 5.3 Traversing a Search Tree with a Schema

Completeness is usually sacrificed in schema-based problem solving. That is, the problem solver does not fully search the entire problem space. The hypothesis is that, although it cannot solve as many problems as a complete solver, the constrained solver will perform substantially better on the problems it can solve. The loss of completeness is unavoidable, since on problems of non-trivial size, an exhaustive search is not computationally feasible. One strength of explanation-based learning is that it can be used to produce new schemata for problem solving, as seen in the last chapter, thereby bringing more problem classes within the scope of the constrained problem solver.

The next two sections describe how **Physics 101** performs mathematical problem solving. First, an equation with which to work is chosen. Following that, the equation is rewritten until the desired information can be extracted.

5.2. Choosing the Initial Equation

An important part of solving a mathematical problem is determining an equation with which to begin. This can be viewed as characterizing the problem [Chi81, Larkin80]. For example, a new physics problem can be viewed as one to be attacked using conservation of momentum, conservation of energy, or by resorting to "first principles" (Newton's laws). This section describes the process by which **Physics 101** chooses the initial equation from which an attempt to solve the problem is made.

If the *unknown* is more complicated than simply an isolated variable, the next phase of the problem solver is entered, where an attempt to eliminate the obstacles in the unknown is made. However, when the unknown is only an isolated variable, an initial equation is chosen. After an equation is chosen, it is algebraically rearranged so that all of the obstacles appear on the right:

hand side of the equation. If this cannot be accomplished, another initial equation is chosen. An alternative to this approach would be to try to rewrite the equation so that only the unknown appeared on the left-hand side of the equation. However, choosing an equation, then rearranging it so that all the obstacles are on the right-hand side, simplifies problem solving because some acceptable variables may remain on the left-hand side.

Only those equations containing the unknown are considered as candidate initial expressions. The equation schemata are divided into two groups: those specifying the initial equations provided to the system (the *first principles*) and those acquired through learning (the *derived concepts*). Furthermore, the derived concepts are sub-divided into two groups: the special and general cases described in the previous chapter. Since the derived concepts combine other equations in previously useful ways, the system attempts to use them first. If the current problem is closely related to a previously-experienced one, then the generalized solution to that problem will lead to a rapid solution of the current problem.

Candidate equations are evaluated in the following order (recall that at each step only those equations containing the unknown are considered):

- (1) *The special-case derived concepts.* The problem solver determines if the preconditions of these schemata are satisfied by the current problem description. Special-case equations are preferred over general ones, as the special-case schemata contain a substantial amount of compiled problem solving. If one of the special cases applies, it is likely it will lead to rapid solution of the problem.
- (2) *The general-case derived concepts.* The system again determines if the preconditions of these schemata are true in the current situation. More problem solving may be needed here than when a special case is used, but there should be substantial savings over resorting to first principles.
- (3) *The special-case derived concepts — revisited.* This time a version of the closed-world hypothesis [Reiter78] is used to make assumptions. Making assumptions can transform an intractable problem (due to complicated mathematics beyond the scope of the problem solver) or insoluble problem (due to insufficient data) into one that can be solved. Assumption-making is done in the following manner. Each precondition of the current special case schema is checked. If any precondition can be proven false (i.e., its negation can

be proven), the current principle is rejected, otherwise it is accepted. This means that any precondition that cannot be shown to be false is assumed to be true. For example, one precondition of some derived concept may state that every external force be zero. If a non-zero external force is discovered, the concept cannot be used. However, if nothing is known about the external forces, the concept can be used. If **Physics 101** makes the closed-world assumption, when providing the final answer, it also lists the assumptions made (those preconditions that cannot be proven true or false, and are assumed to be true).²

- (4) *The first principles.* Assumptions are not made about first principles. All of their preconditions must be provably true.

The issue of making consistent assumptions during mathematical problem solving is an important research issue. The topic of assumption-making is only briefly investigated in **Physics 101**. This research ignores issues involved in ensuring that assumptions made in one step of problem solving are consistent with the rest of the steps. Because special-case schemata contain compiled problem-solving knowledge, when they are chosen as the initial equation they usually lead to a solution after only a small amount of algebraic manipulation. For this reason, not checking consistency is less likely to lead to problems. However, this is not the case when general-case schemata and first principles are used and, hence, assumption-making is not performed when these schema types are evaluated.

The problem of proving preconditions can be formidable. For example, to determine if an external force is zero may require reasoning about all the consequences of Newton's laws. In **Physics 101**, the module that proves preconditions is very limited. It does not combine equations in order to answer queries. It only retrieves assertions made when the problem is described, performs arithmetic, and interprets the connectives and quantifiers of predicate calculus (e.g., *and*, *or*, *not*, *implies*, *for all*, and *there exists*).

Once an equation is chosen, it is algebraically manipulated to bring all of the obstacle variables to the right-hand side. This is done in a manner similar to Bundy's *attraction*.

² Although making assumptions may lead to incorrect answers during problem solving (due to the assumptions being wrong), learning that results from analyzing the solution will not produce incorrect results. This is because the assumptions made are incorporated into the learned schema as preconditions.

collection, and *isolation* methods [Bundy81, Bundy83]. If this process fails, the equation is rejected. These same techniques are used to extract the answer to the problem from the final, obstacle-free equation.

5.3. Transforming an Expression into an Acceptable Form

The problem-solving model used to transform the chosen initial equation into an acceptable form, one without obstacles, is discussed in this section. It is intended to be a psychologically-plausible model of problem solving [Shavlik86b]. Section 5.4 describes the model's relationship to other problem solvers for mathematical domains and its relevance to the difference between human novices and experts.

A novice human problem solver attacks a problem in one of two ways. He may immediately notice a way to progress toward the solution. Alternatively, he may flounder around performing legal, but aimless, operations in an attempt to transform the problem into a familiar form. An expert, on the other hand, can perform in a qualitatively different manner. If the solution is not immediately apparent, he can focus his efforts in a much more guided way. Instead of simply thrashing around, an expert has an appreciation of what kinds of transformations are likely to change the current problem into a soluble problem.

Consider the problem of evaluating the expression

$$M_1 V_1 + M_2 V_2$$

when the values of M_1 and M_2 are known, but those of V_1 and V_2 are not. This expression cannot be evaluated directly. A valid approach might be to substitute equivalent expressions for the unknowns. A novice might perform the unappealing substitutions of figure 5.4. While these transformations are valid, they are unlikely to yield a solution. An expert will appreciate this, and be more likely to perform the pleasing problem transformations of figure 5.4. In this example, there is something about the parallel structure of the problem that makes a parallel substitution more appealing. Yet parallel substitutions are not always aesthetically appealing. For example, consider the substitutions of figure 5.5. In this example, parallel substitution using Newton's third law (every action has an equal and opposite reaction) misses a useful variable cancellation. If only one instantiation of this formula is used, the two forces can be eliminated from the calculation.

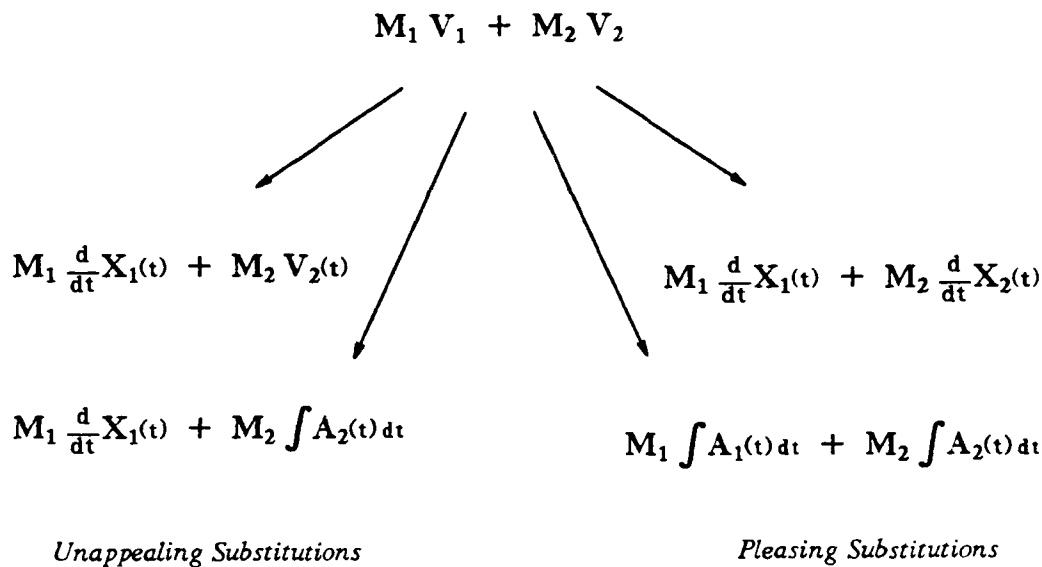


Figure 5.4 Sample Mathematical Substitutions

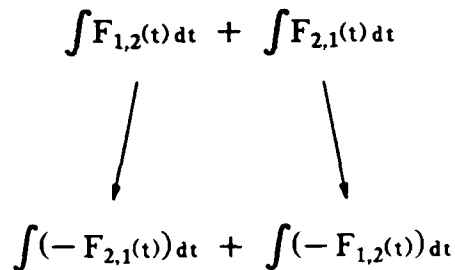


Figure 5.5 Inappropriate Parallel Substitutions

Three qualitatively different strategies for problem solving are incorporated in the model. Attention is selectively focussed according to one of these strategies. *Strategy 1* is hill climbing. Attention is focussed on this strategy as long as some schema moves the problem solver closer to its goal. Occasionally the problem solver will reach a local maximum; there will be no way to move closer to the goal. At these times, the problem solver must diverge from its goal, in the hopes of transforming the current situation into one where hill climbing can again occur. There are two qualitatively different ways by which the problem solver re-focusses its attention during this divergent phase. First, it attempts to transform the current situation in some seemingly useful way. Characteristics that are believed to be of general or domain-specific

problem-solving importance (e.g., symmetry) are to be maintained or introduced; introduction of troublesome characteristics is to be avoided. Such motivated diversions compose *strategy 2*. As a last resort, the problem solver merely selects an arbitrary legal schema. This can lead to aimless floundering, due to the large number of possible combinations of operator sequences. This unmotivated application of schemata is termed *strategy 3*.

Figure 5.6 schematically presents an overview of the three strategies. Strategy 1 is used as long as progress toward the goal is achieved. When a local maximum is reached without the goal being satisfied, a place to jump in the space is needed. Strategy 2 preserves important characteristics of the current state. It is likely that under strategy 2 the problem solver transfers to a new high spot in the problem space. When strategy 2 cannot contribute to the solution, strategy 3 leads to an arbitrary location. Most likely this will be further from the goal, but progress to the goal can commence again.

Under each of the strategies, both derived concepts and the first principles can be used. In this phase of problem solving, these two categories are not treated differently and assumptions are not made. The next sections give examples of the use of these three strategies in *Physics 101*.

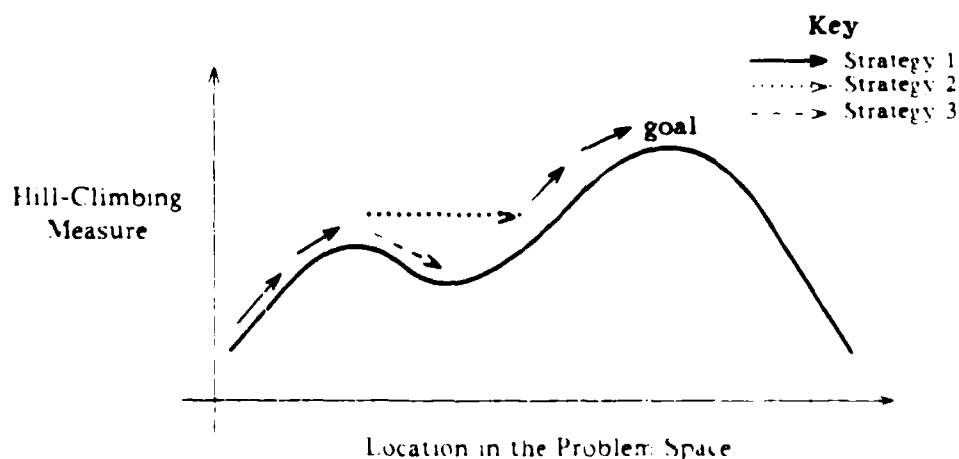


Figure 5.6 The Problem-Solving Model as Hill Climbing

Strategy 1 — Definite Progress

In the **Physics 101** system, the goal is to produce an expression that only contains variables that satisfy some property (e.g., being constant with respect to time, having a known value, etc.). In this case the hill-climbing measure is the number of variables in the expression that do *not* satisfy the property.³ That is, this number is the size of the obstacle set.

Physics 101 contains two basic techniques for reducing the number of undesirable variables. In the first technique, undesirable variables can be replaced using known equations if these formulae only introduce acceptable terms or lead to the cancellation of unacceptable ones (obstacles). An example of this technique is shown in figure 5.7 (discussed below). In the second technique, values of variables can be used if doing so reduces the hill-climbing measure. If the current expression is $A * B * C$, and A 's value is zero, both B and C can be cancelled by replacing A with its numerical value. Similarly, given the expression $A * B * C - B * C$, where A equals one, a numerical replacement can lead to a reduction in obstacles.

A problem-solving schema applied under strategy 1 is illustrated in figure 5.7. The goal is to evaluate the top expression, but the values of the two inter-object forces ($F_{1,2}$ and $F_{2,1}$) are not known. A "substitute-to-cancel-obstacles" schema can detect that the two inter-object

Consider the Expression

$$\int F_{1,2}(t) dt + \int F_{2,1}(t) dt$$

Choose a Variable-Cancelling Substitution

$$\int (-F_{2,1}(t)) dt + \int F_{2,1}(t) dt$$

Bring Cancellers Together

$$\int (-F_{2,1}(t) + F_{2,1}(t)) dt$$

Cancel Variables

$$\int 0 \frac{kgm}{s} dt$$

Figure 5.7 Example of an Operator Applied under Strategy 1

³ Strictly speaking, since it is desired to minimize this measure, "valley descending" is being performed.

forces cancel due to Newton's third law. The schema first applies Newton's third law to replace one of the inter-object forces. It then brings these potential cancelling terms into a position where cancellation can take place. This requires that the two integrals be combined. Finally, the two troublesome variables can be eliminated.

Strategy 2 — Motivated Diversions

There are five techniques used in **Physics 101** that follow the second strategy. They are presented below, in the order they are preferred by the system.

- (1) *Elimination of acceptable variables.* Even when it is not possible to reduce the number of unacceptable terms in an expression, it is a good idea to cancel terms, even acceptable ones. Eliminating these terms may allow productive cancellations that had been prevented by their presence. For example, suppose the problem solver possesses the following formulae, and A and C are acceptable.

$$A = B / C \quad \text{and} \quad B = -D$$

If the current expression is $A * C + D$, the problem solver cannot reduce the number of obstacles. However, if the first of the above formulae is used, some terms can be cancelled, although the number of obstacles is momentarily increased.

- (2) *Elimination of integrals and derivatives.* Performing calculus is harder than performing algebra. When it is not possible to eliminate terms, it is often a good idea to eliminate calculus structure. An example where **Physics 101** removes calculus from an expression is shown in figure 5.8. Here the program detects that it can eliminate the derivative because it knows the derivatives of all the terms being differentiated. Applying this operator adds four steps to the calculation sequence. After this schema is applied, direct progress toward the goal state can be made. (Continuing the calculation in figure 5.8 leads, as discussed in section 7.1, to the principle of conservation of energy.)

- (3) *Preservation of expression type.* Assume the system has the following two equations.

$$(i) \ A = (D * E) \quad (ii) \ A = (F + G)$$

If the current expression is $A * B * C$, equation (i) would be preferred as this would

$$\begin{aligned}
& \frac{d}{dt} \left(\frac{1}{2} M_1 V_{1,y}^2(t) + M_1 g X_{1,y}(t) \right) \\
\text{SeparateDerivatives} &= \frac{d}{dt} \left(\frac{1}{2} M_1 V_{1,y}^2(t) \right) + \frac{d}{dt} (M_1 g X_{1,y}(t)) \\
\text{ConstsOutOfDerivatives} &= \frac{1}{2} M_1 \frac{d}{dt} V_{1,y}^2(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
\text{Differentiate} &= \frac{2}{2} M_1 V_{1,y}(t) \frac{d}{dt} V_{1,y}(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
\text{RemoveIdentities} &= 1 M_1 V_{1,y}(t) \frac{d}{dt} V_{1,y}(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
&= M_1 V_{1,y}(t) \frac{d}{dt} V_{1,y}(t) + M_1 g \frac{d}{dt} X_{1,y}(t)
\end{aligned}$$

Figure 5.8 An Application of the Substitute-Calculus Schema

maintain the property that the expression is a product of terms. Conversely, given $A + B + C$, the second equation is preferred, because after substitution the expression continues to be a sum of terms. There is a strong reason for preserving expression type, one involving more than aesthetics. In the first example the system can produce

$$D * E * B * C \quad \text{or} \quad (F + G) * B * C.$$

In the result on the left, all the terms are equally accessible. Future substitutions involving B , for example, can cancel D or E . (Recall that elimination of undesirable variables is the mechanism that leads to the goal state.) The right-hand result requires that a replacement for B cancel F and G together.

- (4) *Preservation of structural symmetry.* When similar additive or multiplicative structure is present, the same general rule should be used repeatedly whenever possible. For example, given the following expression, substitutions involving all three of the A 's or all three of the B 's would be favored.

$$A_1 B_1 + A_2 B_2 + A_3 B_3$$

This accounts for the unpleasant quality of the first transformations of figure 5.4. It would be better to replace both of the velocities either by the derivatives of position or by the integrals of acceleration. A mixture is not appealing.

- (5) *Creation of a Same Type Expression.* It generally is a good policy to reason with variables, rather than numbers. This is especially important if the solution is to be analyzed in order to determine its generality. However, if the replacement of a variable with its value creates a "same type" expression, the replacement should occur under strategy 2. For example, consider the following expression, where $C = 0$. Replacing C with zero and simplifying creates a multiplicative expression. The variable B would become accessible to combination with variables A and D . Similarly, an additive expression can often be produced when the value of some variable, embedded in a multiplicative sub-expression, is 1.

$$A (B + C) D$$

Strategy 3 — Floundering Around

In strategy 3, **Physics 101** looks for a legal substitution and applies it. Only one substitution is made in the current equation, in order to minimize this undirected perturbation of the calculation. Also, if after using strategy 3, strategies 1 and 2 are still not applicable, the current search path terminates. That is, strategy 3 is not used in successive calculation steps. This is done to prevent the problem solver from traversing the entire (possibly infinite) search space and is one of the principle reasons the solver is incomplete. Upon failing to produce a solution on its own accord, **Physics 101** requests one from its teacher. If a new schema results from analyzing the solution, the system will be able to solve the current intractable problem, and related problems, in the future.

Choosing Among Alternatives

When several schemata are seen as being equally viable, the problem solver must choose which to apply. Under strategy 3 the choice is made arbitrarily. In the other two cases, the choice is made using the second strategy. For example, if there are a number of ways to cancel two obstacles, a way that preserves the symmetry of the situation is preferred. When the second strategy does not distinguish a single choice, the schema that introduces the smallest number of new variables is used. If there still is no clear favorite, a random choice is made. Choice points are maintained for decisions made under strategies 1 and 2. The system

backtracks to the most recent choice point when no further progress can be made along the current search path.

The system does not record *all* possible choices, rather *only* those possibilities suggested by the highest ranking strategy are kept. That is, if there are multiple choices under strategy 2, **Physics 101** never considers strategy 3 at that point, even if all of the alternatives under strategy 2 fail. This contributes to the efficiency of the problem solver. It is also a source of incompleteness.

5.4. Summary

This section describes the overall task accomplished by **Physics 101**'s problem solver. It discusses how it chooses equations with which to begin work and describes how these chosen equations are manipulated into an acceptable form. Section 6.3 presents further details on the problem solver, including a description of the structure of problem-solving schemata and a discussion of how schemata are activated and de-activated during problem solving.

There are several ways the initial equation is chosen. Equations derived by combining other equations are preferred over first principle equations. Special-case equation schemata are preferred over general ones, as the special cases contain a substantial amount of compiled problem solving. Also, the closed-world assumption is used before resorting to the use of first principles. Here assumptions are made wherever preconditions to a learned schema cannot be shown to be false. Making assumptions can transform an intractable or insoluble problem into one that can be solved. Once an initial equation is chosen, all of the obstacles in it are transferred to the right-hand side, and the second phase of problem solving is entered. In this phase, the system attempts to eliminate all of the obstacles.

Three qualitatively different strategies have been developed that help focus attention during the second phase of problem solving. The first strategy is to apply schemata that will lead to definite progress toward the goal. Attention will be focussed by this strategy as long as some schema in this class is applicable. When clear progress can not be achieved, the problem solver must decide how best to proceed. It then invokes the second strategy to select schemata that preserve possibly important characteristics of the current problem. These schemata are likely to keep the problem solver from diverging sharply from the goal while possibly generating

the application of schemata by the first strategy. When the problem solver can follow neither of the first two strategies, it arbitrarily applies equation schemata.

Other mathematical reasoning research can be viewed in terms of these strategies. Bundy's meta-level solution methods [Bundy81, Bundy83] follow strategy 1. He considers solving complicated equations containing a single unknown variable (there may be multiple occurrences of the unknown, however). His *attraction*, *collection*, and *isolation* methods always bring one closer to the goal of having the only occurrence of the unknown isolated on the left-hand side of an equation. The mathematical problem-solving methods learned by Silver's **LP** program [Silver86] follow strategy 2. **LP** acquires information that constrains the choice of applicable operators. The learned operators are not guaranteed to bring the problem solver closer to a final solution. The **LEX** system of Mitchell [Mitchell83b] acquires heuristics that estimate when it is wise to apply an integration operator. It learns how operators previously used only under strategy 3 can be applied by strategy 2. The later **LEX2** system [Mitchell83a] learns under what conditions a specific integration operator will lead to a solution. This can be viewed as learning how to apply, under strategy 1, an operator previously used only under strategy 3.

Several other approaches to learning about problem solving can be analyzed with respect to these categories. Iba's approach [Iba85] to determining useful subsequences of operators (those form which macro-operators should be constructed) is to use a "peak-to-peak" heuristic. These peaks are local maxima of the evaluation function used to guide the search for the solution to the current problem, and sequences are segmented at the peaks. The resulting macro-operators fall under strategy 1 if they move from one peak to a higher one. If the result is an operator sequence that leads away from the goal, it falls under strategy 2, because it is probably better to jump to another relatively high point than to make an arbitrary move. Minton, in his **MORRIS** system [Minton85], also focusses on deciding which macro-operators to learn. One type, called *T-macro*s, are learned when an operator sequence first appears to be moving away from the goal but by the end of the sequence progress toward the goal has been achieved. These macro-operators fall under strategy 1.

Finally, these three strategies can be viewed in terms of simulated annealing problem-solving models [Hinton86, Kirkpatrick83]. Progress using strategy 1 involves the movement toward a problem space minima. Strategy 2 is analogous to slightly increasing the system "temperature" when stuck at a local minima that is not a goal state. Here it is desired that the

problem solver does not drift too far in the problem space. Strategy 3 potentially involves much greater increases in system temperature, and, hence, much greater jumps in the problem space.

The second strategy illustrates an essential difference between novice and expert problem solvers [Chi81, Chi82, Larkin80, Schoenfeld82]. It is easy to recognize definite progress toward a goal and it is easy to recall which operators can be legally applied. Expertise involves knowing which characteristics of a situation should be preserved (or created) when no way to definitely progress toward the goal is known.

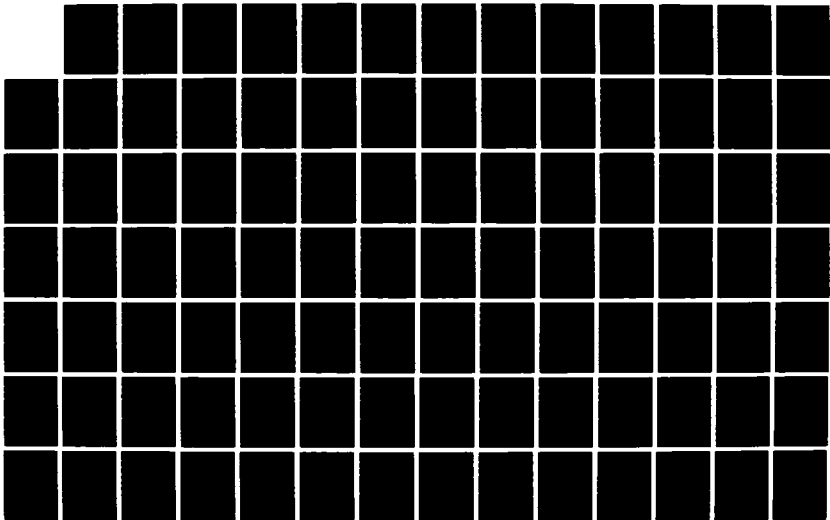
NO-A191 327

GENERALIZING THE STRUCTURE OF EXPLANATIONS IN
EXPLANATION-BASED LEARNING(U) ILLINOIS UNIV AT URBANA
COORDINATED SCIENCE LAB J M SHAYLIK DEC 87
UNCLASSIFIED UIU-ENG-87-2276 N00014-86-K-8309

2/4

F/G 23/2

ML





1.0

2.8



2.5

3.15

3.6

4.0

4.5

2.0



1.1

1.8



1.25



1.4



1.6

Chapter 6

Algorithmic Details of Physics 101

Details of **Physics 101**'s algorithms and data structures are presented in this chapter. Discussed are how the cancellation graph is constructed, how this graph is used to build the generalized explanation structure, and how problem-solving schemata are selected.

It is often necessary to distinguish between different occurrences of the same mathematical variable¹ (e.g., F , M , and A). For example, there are two mass terms in line 2 of figure 3.5. One first appeared in the left-hand side of the calculation, while the other arose from the $A = F_{net} / M$ formula substitution. In the implementation, every variable in a calculation maintains pointers to the variable that produced it and, if it is replaced in a formula substitution, to all of its direct descendants. In addition every variable replaced in a substitution records the specific and general forms of the equation used.

¹ Unfortunately the word *variable* has at least three connotations. There are the variables used in mathematical equations, those used in predicate calculus, and those used in programming languages. The word *variable* will always refer to mathematical variables unless preceded by modifiers.

Physics 101's algorithms are expressed here in a pseudo-code, designed for readability. (The actual implementation is written in Lisp.) In these algorithms, the notation *record.field* is used to represent a given field of a record, and back arrows (\leftarrow) indicate value assignment. The construct

for each element in set unless test do statement

means that *element* is successively bound to each member of *set* and, unless *test* is true, *statement* is evaluated.

6.1. Constructing the Cancellation Graph

The cancellation graph for a calculation describes how the calculation supports the problem's solution. This graph records how unacceptable variables (those that may not appear in the final right-hand side of a calculation) are eliminated from the calculation. Of particular interest are those unacceptable variables descended from the problem's unknown (via rewriting using equation schemata). How these cancellations are set up and how any undesirable side-effects of the cancellations are circumvented are recorded. This process determines the role of each variable appearing in the left-hand side of the calculation. Irrelevant variables are detected and are not used during the calculation reconstruction phase in which the calculation is generalized.

Figures 6.1 and 6.2 contain the algorithm for constructing cancellation graphs, while table 6.1 lists the fields referenced and describes their setting when the algorithm commences. (Fields ending with question marks are boolean-valued.) Further details on these fields are provided as the algorithm is discussed.

A sample problem is used to illustrate the algorithm. It is contained in table 6.2, and its cancellation graph appears in figure 6.3. The features of this specific cancellation graph are explained as the algorithm is described. Recall that the first expression in a sequence of calculation steps is called the *left-hand side* of the calculation, while the subsequent expressions are termed *right-hand sides*.

In the sample calculation, Greek letters are used for variables that cannot appear in the final right-hand side of the calculation. For example, it may be that the final right-hand side can only contain variables whose value is known or variables that are time-independent. The goal is to rewrite the left-hand side, via a series of equality-preserving transformations, into an

```

procedure CheckForObstacles (variable)
    if variable.notChecked? then
        variable.notChecked?  $\leftarrow$  false
        variable.obstacleSet  $\leftarrow$  FindObstacleSet({variable})
        AnalyzeObstacles(variable.obstacleSet)

procedure FindObstacleSet (variables)
    answer  $\leftarrow$  { } /* answer is a local variable. */
    for each variable in variables unless variable.eliminated?
        do if variable.replaced?
            then answer  $\leftarrow$  answer  $\cup$  FindObstacleSet(variable.descendants)
            else if variable.unacceptable?
                then answer  $\leftarrow$  answer  $\cup$  {variable}
    return answer

procedure AnalyzeObstacles (obstacleSet)
    for each obstacle in obstacleSet do Eliminate(obstacle)
    for each obstacle in obstacleSet do CheckForSecondaryObstacles(obstacle)

procedure Eliminate (variable)
    variable.eliminated?  $\leftarrow$  true
    AnalyzeBlockers(variable.blockers)
    AnalyzePartners(variable.partners)
    AnalyzeCancellers(variable.cancellers)

```

Figure 6.1 Building the Cancellation Graph - Part I

```

procedure AnalyzeBlockers (blockers)
    for each blocker in blockers unless blocker.eliminated? do Eliminate(blocker)

procedure AnalyzePartners (partners)
    for each partner in partners unless partner.eliminated?
        do partner.eliminated?  $\leftarrow$  true
        AnalyzeBlockers(partner.blockers)

procedure AnalyzeCancellers (cancellers)
    for each canceller in cancellers
        do canceller.eliminated?  $\leftarrow$  true
        AnalyzeBlockers(canceller.blockers)

procedure CheckForSecondaryObstacles (variable)
    for each blocker in variable.blockers do CheckForSecondaryObstacles(blocker)
    for each partner in variable.partners
        do CheckForObstacles(partner.producer)
        for each blocker in partner.blockers do CheckForSecondaryObstacles(blocker)
    for each canceller in variable.cancellers
        do CheckForObstacles(canceller.producer)
        for each blocker in canceller.blockers do CheckForSecondaryObstacles(blocker)

```

```

CheckForObstacles(unknown)                                /* Construct the cancellation graph. */

```

Figure 6.2 Building the Cancellation Graph - Part II

Table 6.1 Fields Used in the Cancellation Graph Algorithm

Field	Initial Setting
<i>notChecked?</i>	initially <i>true</i> , reset in algorithm when checked for obstacles
<i>obstacleSet</i>	initially {}, set in algorithm
<i>unacceptable?</i>	<i>true</i> if this variable cannot appear in the final right-hand side of the calculation
<i>eliminated?</i>	initially <i>false</i> , set in algorithm
<i>replaced?</i>	<i>true</i> if replaced by a rewrite rule during the calculation
<i>descendants</i>	variables introduced when this variable rewritten
<i>blockers</i>	set of variables preventing cancellation of this variable
<i>partners</i>	set of variables in the cancelled expression in addition to this variable
<i>cancellers</i>	set of variables that cancelled this variable
<i>producer</i>	variable in the left-hand side of the calculation that produced this variable

expression containing no Greek letters. The equation schemata used to produce each new expression are shown in the right column. A mixture of domain-specific and general algebraic rewrite rules are used. β is the variable about which some information is sought, i.e., it is the problem's unknown. In this example, subscripts are used to differentiate various instantiations of the *same* variable (e.g., b_1 and b_2). In other words, b_1 and b_2 both refer to the variable b .

This sample calculation illustrates the major components and attributes of cancellation graphs. The graph specifies the role of each variable in the left-hand side of the calculation. Combined together in the given manner, these variables support the answer to the question about the unknown. One important feature is that unnecessary variables (e.g., α in table 6.2) do not appear in the graph.

The first step in building a cancellation graph is to create the *obstacle set* for the problem's unknown. (This obstacle set is called the *primary obstacle set*. Obstacle sets constructed in later phases of the algorithm are called *secondary sets*.) An obstacle set for a collection of variables consists of those variables possessing the following properties:

Table 6.2 Sample Calculation	
Calculation Steps	Rewrite Rules Used
$\alpha ((a - b_1) \frac{\beta}{c_1} + d) g$	β is the unknown
(1) $= f ((a - b_1) \frac{\beta}{c_1} + d) g$	$\alpha = f$
(2) $= f ((a - b_1) \frac{\delta_1 c_2}{c_1} + d) g$	$\beta = \delta c$
(3) $= f ((a - b_1) \delta_1 + d) g$	$?x / ?x = 1$ $?x 1 = ?x$
(4) $= f ((\epsilon_1 + b_2 - b_1) \delta_1 + d) g$	$a = \epsilon + b$
(5) $= f (\epsilon_1 \delta_1 + d) g$	$?x - ?x = 0$ $?x + 0 = ?x$
(6) $= f (\epsilon_1 \delta_1 + \gamma_1 - \epsilon_2 \delta_2) g$	$d = \gamma - \epsilon \delta$
(7) $= f \gamma_1 g$	$?x + ?y = ?y + ?x$ $?x - ?x = 0$ $?x + 0 = ?x$
(8) $= \frac{f \gamma_1 h}{\gamma_2}$	$g = \frac{h}{\gamma}$
(9) $= f h$	$?x / ?x = 1$ $1 ?x = ?x$

- (1) they are in the given collection or descended (via rewriting with equation schemata) from a member of it.
- (2) they cannot appear in the final right-hand side of a calculation.
- (3) they are cancelled in the calculation, and
- (4) they are not marked earlier in the algorithm as being eliminated from consideration.

The function *FindObstacleSet* assumes that the provided calculation is successful. This means that if an unacceptable variable is not replaced, it must be cancelled, since there can be no unacceptable variables in the final right-hand side of a calculation.

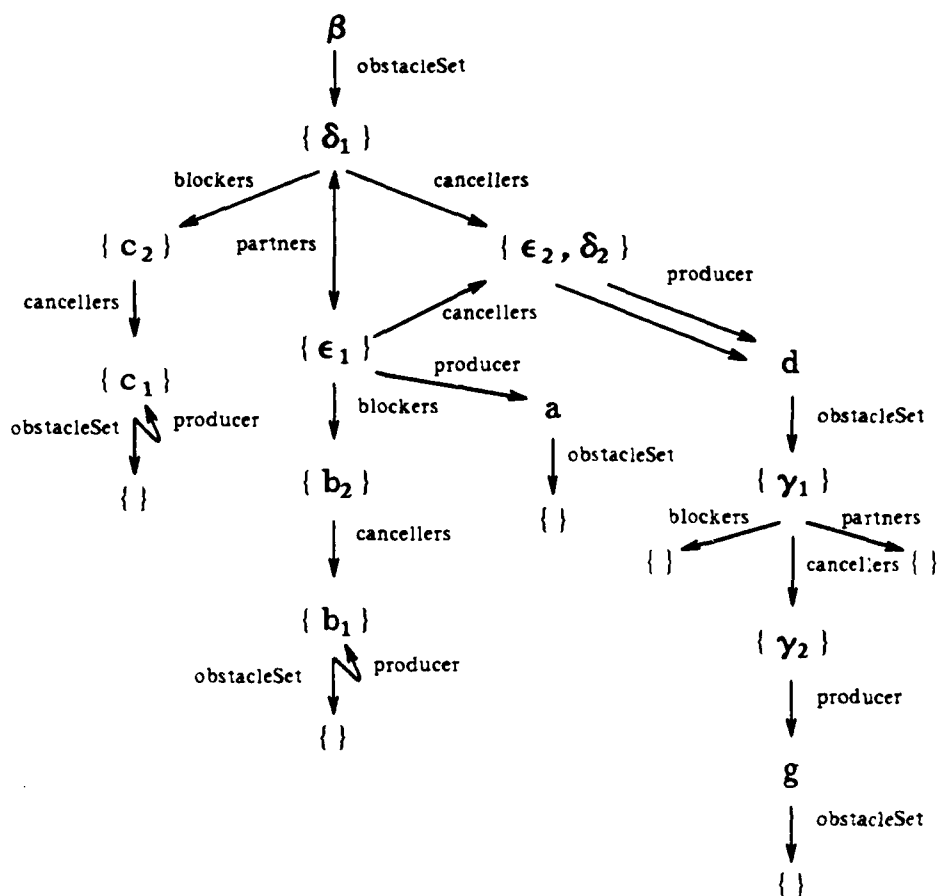


Figure 6.3 The Cancellation Graph for the Calculation in Table 6.2

The only member of the sample problem's primary obstacle set is δ_1 . It appears in line 2 and is cancelled in line 7 of table 6.2. The only other descendant of the unknown β is c_2 , and c_2 can appear in the final expression of the calculation. The next things to do are see how the obstacle is cancelled and then determine if there are any undesirable side-effects of doing this. If there are side-effects, how they are circumvented in the specific example needs to be determined.

An obstacle set is analyzed once it is constructed, and how each element of this set is eliminated from the calculation is determined. This involves three steps.

- (1) Determining how those variables blocking the cancellation are eliminated.
- (2) Determining how the obstacle's cancellation partners are brought into position for the cancellation.
- (3) Determining how the cancellers of the obstacle are brought into position.

Analyzing the Blockers of the Obstacle

The first step is to determine how the obstacle is isolated for cancellation. When introducing an obstacle into a calculation (via the application of equation schemata), additional variables are often unavoidably also introduced. Some of these may be blocking the cancellation of the obstacle, and, if so, how these *blockers* are eliminated needs to be ascertained.

The blockers of an obstacle depend on the manner by which the obstacle is cancelled. This is illustrated in table 6.3. The *blocker sets* for various expressions and cancellation types are shown. The first column contains the expression descended from the producer of the obstacle, while the second column contains the full expression containing the obstacle that is cancelled when the obstacle is eliminated. For example, if the equation schemata used to cancel the obstacle a_1 specifies $\frac{a_1 + b_1}{a_2 + b_2} = 1$, then the cancelled expression is $a_1 + b_1$. (In this case b_1 is the only cancellation partner and $\{a_2, b_2\}$ is the canceller set.) In the last two columns are the blocker sets as a function of the cancellation type.

In table 6.3's first example, obstacle a is *additively isolated* and *multiplicatively blocked*, while in the second and third examples the opposite is true. The fourth and fifth lines are examples of expressions that are blocked for both types of cancellation. The last three examples illustrate the effect of a more complicated cancelled expression. As shown in line 6, variables in the cancelled expression are not blockers. Line 7 shows that an additive cancellation partner does not effect a sum of variables, while a multiplicative cancellation partner means that the sum of variables must be reduced to a single term (line 8).

Table 6.3 Blockers of Obstacle a

	Expression Descended from a 's Producer	Full Cancelled Expression	Obstacle Cancellation Type	
			Additive	Multiplicative
(1)	$a + b + c$	a	$\{\}$	$\{b, c\}$
(2)	$a b c$	a	$\{b, c\}$	$\{\}$
(3)	$a (b+c) d$	a	$\{b, c, d\}$	$\{\}$
(4)	$(a + b) c + d$	a	$\{c\}$	$\{b, d\}$
(5)	$(a b + c) d$	a	$\{b, d\}$	$\{c\}$
(6)	$a + b + c$	$a + b$	$\{\}$	$\{c\}$
(7)	$a + b + c$	$a + x$	$\{\}$	$\{b, c\}$
(8)	$a + b + c$	$a x$	$\{b, c\}$	$\{b, c\}$

In the calculation of table 6.2, the obstacle δ_1 is additively cancelled (line 7) and has a multiplicative cancellation partner (ϵ_1). The variable c_2 is the only member of δ_1 's blocker set.

Analyzing the Partners of the Obstacle

Next, the cancellation partners of the obstacle are analyzed. This entails determining how the blockers of each partner are eliminated. By eliminating these blockers, the partners are positioned so they can participate in the cancellation with the obstacle. In table 6.2, the only blocker of ϵ_1 is b_2 .

Analyzing the Cancellers of the Obstacle

Finally, the cancellation of the obstacle must be understood. This requires determining how the blockers of the cancellers are eliminated in order to properly position the cancellers so they can eliminate the obstacle. The expression $-e_2 \delta_2$ cancelled the only member of β 's obstacle set in line 7 of table 6.2. No variables blocked this cancellation.

Checking for Secondary Obstacles

Once all the members of an obstacle set are eliminated, a check is made (in the procedure *CheckForSecondaryObstacles*) to see if this process lead to the existence of additional

(secondary) obstacle sets. Cancelling an obstacle requires the presence of its partners and cancellers, as well as those variables that cancelled the blockers of the obstacle, the blockers of its partners, and the blockers of its cancellers. Hence, additional variables must be present in the left-hand side of the calculation. These *producer* variables are checked to see if their presence means additional obstacle sets are present in the calculation. If so, these secondary obstacle sets are analyzed in the same manner as the primary obstacle set.

Notice that eliminating the secondary obstacles can also require additional variables in the left-hand side of the calculation. These variables may also have obstacle sets (again called secondary sets) that need to be eliminated. Since the successful calculation contained a finite number of calculation steps (and, hence, a finite number of potential obstacle variables), the algorithm always terminates.

There are several cancellers in the calculation of table 6.2: c_1 , b_1 , ϵ_2 , and δ_2 . The first two of these appear in the left-hand side of the calculation (and, hence, are their own producers). The third and fourth cancellers are produced by d . The obstacle set of d only contains γ_1 , which is introduced in line 6. Although not acceptable in the final right-side of the calculation, variables ϵ_2 and δ_2 are not members of this obstacle set because when this set is constructed these variables are already eliminated. The variable g in the left-hand side of the calculation produces the canceller of γ_1 and the obstacle set for g is the empty set.

The producers of cancellation partners are also checked to see if they have obstacle sets. The variable a produced ϵ_1 , the cancellation partner of δ_1 . The obstacle set of a is also empty.

Notice that all of the variables, other than α , in the left-hand side of table 6.2's calculation appear in figure 6.3. This graph records the role of each variable in the determination of the desired property of the unknown. The variable α does not appear in the cancellation graph because it serves no purpose in the achievement of the goal. One of the important features of the cancellation graph process is that irrelevant variables in the left-hand expression can be detected. This is one manner in which the initial explanation can be altered.

6.2. Using the Cancellation Graph

Once the cancellation graph is constructed, it can be used to do such things as explain why the solution technique worked and how it generalizes. This section describes the algorithm that uses

the cancellation graph to construct an augmented explanation structure, one that leads to the general equation schema underlying the sample calculation.

During this process the sample calculation is redone, this time in the general case, using the general versions of all equation schemata. Unacceptable variables in the specific problem are considered unacceptable in the general case and acceptable variables in the specific case are considered acceptable in general. The obstacle graph provides the guidance as to which variables are unacceptable and how they can be eliminated. Thus, the algorithm relies on the assumption that the specific situation is representative of future situations, a fundamental, but often implicit, assumption in explanation-based learning.

Figures 6.4 and 6.5 present the algorithm for reconstructing the calculation. This algorithm is very similar to that used for building the cancellation graph. Obstacles and their blockers, partners, and cancellers are again the focus. Basically, the algorithm works as follows. First, the general version of the unknown is used as the initial left-hand side of the calculation, and then the general versions of the equation schemata used in the specific calculation are applied until the primary obstacles are introduced in their general form.² Next, the general versions of any blockers are eliminated, which may lead to the introduction of more general variables in the left-hand side of the calculation. Following this, any partners of the obstacle are introduced, also in their general form, and their blockers are eliminated. The general versions of the cancellers are then introduced and their general blockers eliminated. After this, the cancellation is recorded and the obstacles and their cancellers are eliminated from the calculation. Finally, secondary obstacles are recursively analyzed. More details on these steps are provided in the remainder of this section.

The first step in analyzing a general obstacle is to introduce it into the calculation. This involves finding its *producer* in the general case, inserting the producer into the left-hand side of the calculation, then performing the general versions of the substitutions performed in the specific calculation into the general version of the obstacle appears in the general calculation. The same procedure is used to introduce partner and canceller variables into the general calculation.

² The unexpanded forms of equations are used. Recall that in solving the specific problem, indefinite summations (\sum) and products (\prod) are expanded into ordinary sums and products.

```

procedure AnalyzeGeneralObstacles (variable)
    for each obstacle in variable.obstacleSet do IntroduceIntoGeneralCalculation(obstacle)
    for each obstacle in variable.obstacleSet unless OR(obstacle.notUsed? obstacle.eliminated?)
        do EliminateInGeneral(obstacle)
    for each obstacle in variable.obstacleSet unless OR(obstacle.notUsed? obstacle.eliminated?)
        do EliminateSecondaryObstacles(obstacle)

procedure EliminateInGeneral (variable)
    EliminateGeneralBlockers(variable.blockers)
    IntroduceGeneralPartners(variable.partners)
    IntroduceGeneralCancellers(variable.cancellers)
    RecordGeneralCancellation(variable.generalVersion)

procedure EliminateGeneralBlockers (blockers)
    for each blocker in blockers unless blocker.eliminated? do EliminateInGeneral(blocker)

procedure IntroduceGeneralPartners (partners)
    for each partner in partners unless partner.eliminated?
        do IntroduceIntoGeneralCalculation(partner)
        EliminateGeneralBlockers(partner.blockers)

procedure IntroduceGeneralCancellers (cancellers)
    for each canceller in cancellers
        do IntroduceIntoGeneralCalculation(canceller)
        EliminateGeneralBlockers(canceller.blockers)

```

Figure 6.4 Building the General Version of the Calculation - Part I

```

procedure EliminateSecondaryObstacles (variable)
  for each blocker in variable.blockers do EliminateSecondaryObstacles(blocker)
  for each partner in variable.partners
    do AnalyzeGeneralObstacles(partner.producer)
      for each blocker in partner.blockers do EliminateSecondaryObstacles(blocker)
  for each canceller in variable.cancellers
    do AnalyzeGeneralObstacles(canceller.producer)
      for each blocker in canceller.blockers do EliminateSecondaryObstacles(blocker)

procedure IntroduceIntoGeneralCalculation (variable)
  <described in text>

procedure RecordGeneralCancellation (generalVariable)
  <described in text>

```

```

AnalyzeGeneralObstacles(unknown)           /* Reconstruct the calculation. */
SimplifyCalculation()                       /* Simplify the new calculation. */

```

Figure 6.5 Building the General Version of the Calculation - Part II

The producer in the general case may not always be the general version of the specific calculation's producer. Usually the general producer is the general version of the specific producer. However, this is not the case when the obstacle is produced because a *problem-specific* equation is used. When a problem-specific equation is used, the producer of a variable is found by tracing backwards through the application of equation schemata, starting at the one that directly introduced the variable, until the problem-specific equation is found. The producer is the variable, introduced at this step, that is an ancestor of the variable being introduced. For example, if the following three equations are used to produce δ , where the first equation is problem-specific, then the general producer of δ is β .

$$\alpha = \beta$$

$$\beta = \chi$$

$$\chi = \delta$$

If the producer does not appear in the left-hand expression of the specific calculation, its proper form in the left-hand side needs to be determined. This entails passing the producer backwards through all the calculation steps between the one in which the producer is introduced and the initial step. The only place the producer is altered is when it is involved in calculus. When this occurs, the expression that will be converted into the producer by this calculus is determined. For example, if after differentiating, the producer P results, then the following must be solved for P' , the new producer.

$$\frac{d}{d?x} P' = P$$

The result is $P' = \int P d?x$. The reverse transformation holds for integration. An example of this appears in section 7.1, where a problem-specific equation is used in a calculation.

There is one exception to this method for determining producers. Because it is desired that the unknown appear in the left-hand expression, if the specific producer of an obstacle is the unknown, then the general producer is the general version of the unknown, regardless of the equation schemata used to produce the obstacle. This means that a problem-specific equation may be used in the general calculation, and a precondition of the resulting schema is that this problem-specific equation hold in the current situation.

Once the producer of a variable is determined, the next step is to merge it into the current left hand-side of the general calculation. This depends on how it relates to other producers in the current left-hand side. For example, if there are none, it becomes the original left-hand side. If it is the producer of an additive partner, it is added to the producer of the other variable in the partnership. Similarly, if it produces a multiplicative canceller, it divides the related producer.

After the producer of a variable is added to the general calculation, the equation schemata that convert the producer into the variable of interest are applied in their general form. While this is occurring, the necessary unifications among predicate calculus variables in the equation

schema and its preconditions are maintained using the standard explanation-based algorithm EGGS [Mooney86].

Whenever the general version of an equation is added to the general calculation, pointers are maintained between the variables in the general calculation and their specific calculation counterparts. This is illustrated in figure 6.6 for a sample substitution where the internal force on an object is replaced by the sum of inter-object forces.

Even if the general formula that produced it does not contain an indefinite summation or product, a variable may have multiple specific versions. For example, $A_{j,?c}$ in figure 4.3 has $A_{2,x}$ and $A_{3,x}$ as its specific versions. In addition to pointing to each of its specific versions, each variable in the general calculation has its *range* recorded. The range of a variable defines allowable values for subscripts. For instance, the range of $A_{j,?c}$ in figure 4.3 specifies that j

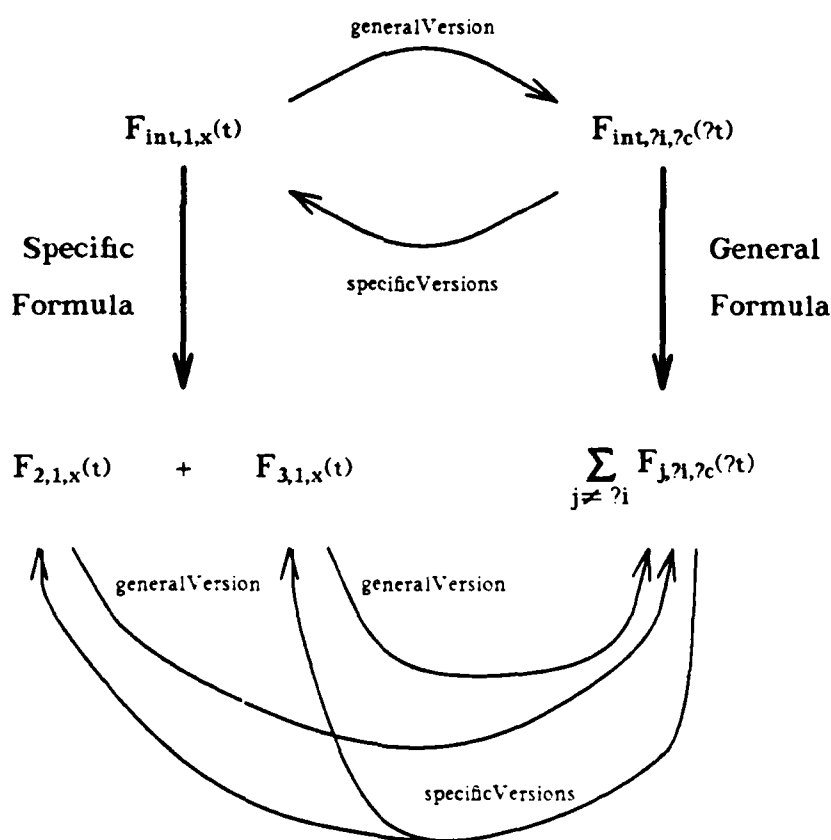


Figure 6.6 Mapping Between Specific and General Variables

ranges over the elements of *ObjectsInWorld*, except for *?s*, and that *?c* can take on a single value.

If there is no specific counterpart for a general variable, the specific example did not address all of the issues inherent in the general calculation. In particular, it did not address how this variable should be handled. Since there is no indication whether or not this general variable is acceptable, generalization terminates and a general equation schema is not learned. An example of this appears in section 7.4, where a two-body collision problem is discussed.

Variable ranges are used to determine the necessary range of the producers of cancellers. If X_j ranges over the elements of *ObjectsInWorld*, except for *i*, then the producer of its canceller receives the same range. Hence, if it is an additive cancellation and *P* is the producer of *X*'s canceller, then $\sum_{\substack{j \in \text{ObjectsInWorld} \\ j \neq i}} P_j$ is merged into the left-hand side of the calculation. For a multiplicative cancellation $\prod_{\substack{j \in \text{ObjectsInWorld} \\ j \neq i}} P_j$ is used.

Once a variable and its cancellers are present in the general calculation the cancellation is recorded. This involves altering the ranges of the variables involved in the cancellation, removing the cancelled possibilities. The necessary unifications are also recorded. If the range of a subscript becomes empty, the general variable is marked as being eliminated. To illustrate this, assume that the range of one instantiation of the variable X_j is the singleton *i*, while the range of its canceller is from 1 to *n*. After the cancellation is recorded, the first instantiation is cancelled and the second has the new range of 1 to *n* except *i*.

After the cancellation is recorded, it is determined whether or not any secondary obstacles are introduced by the elimination of the blocking or obstacle variable. If so, these secondary obstacles are analyzed in a recursive manner. Finally, the resulting calculation is algebraically simplified and any remaining arithmetic is performed. Preconditions of the equation schemata used in the general calculation are collected, with all predicate calculus variables in their constrained form, and a new equation schema is constructed.

6.3. Problem Solving in Physics 101

In this section, more details about the algorithms and data structures of **Physics 101**'s problem solver are presented. Earlier, a hierarchy of some of the system's mathematical

problem solving techniques was presented. That figure is repeated below, as much of this section refers to its contents. The organization of several of the schemata in this figure are described below.

To be used, a problem-solving schema must be *active*. Various schemata activate other schemata, and when an active schema is no longer relevant, it deactivates itself. The main purpose of this activation procedure is to speed up the problem solver. Time is not wasted checking inactive schemata to determine whether or not they are currently relevant. When problem solving commences on a new problem, all existing problem-solving schemata are activated.

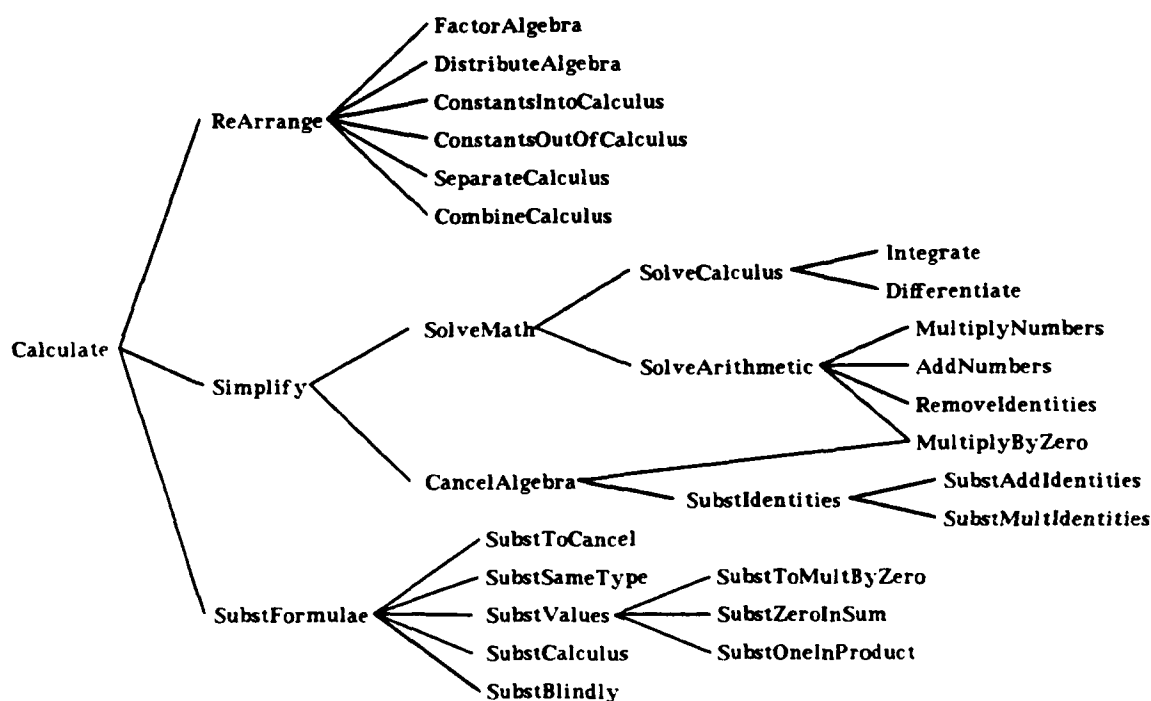


Figure 6.7 Some of the System's Mathematical Techniques (repeated)

Table 6.4 describes the contents of a problem-solving schema. Preconditions to these schemata are divided into two groups. In the first group (called the *keep-active preconditions*) are those preconditions that indicate if the schema should remain active. If these are not satisfied, the schema deactivates itself. The second group of preconditions determine if the schema should be applied at the current stage of problem solving. Note that a schema can currently be inapplicable, but still be relevant. Some of the examples later in this section will illustrate this. The next field in a schema (the *initial activation set*) contains a list of other problem-solving schemata. When both groups of preconditions for a schema are satisfied, the schemata in this list after activated before the schema's body is executed. If *success* is returned when the schema's body is executed, the elements of the *final activation set* are activated. The algorithm for applying problem-solving schemata is presented in figure 6.8.

Table 6.4 The Structure of a Problem Solving Schema

<i>Keep-Active? Conditions</i>	Conditions that are checked to see if this schema should remain active.
<i>Other Preconditions</i>	The remaining preconditions for applying this schema.
<i>Initial Activation Set</i>	Schemata to activate <i>before</i> executing the schema body.
<i>Schema Body</i>	The body of the schema: may describe a transformation of the current expression or be a description of an ordering among other problem-solving schemata.
<i>Final Activation Set</i>	Schemata to activate <i>after</i> executing the schema body.

```
procedure ApplyProblemSolvingSchema (schema)
  if Active?(schema) then
    if Satisfied?(schema.keepActiveConditions) then
      if Satisfied?(schema.otherPreconditions) then
        ActivateTheseSchemata(schema.initialActivationSet)
        if ExecuteBodySuccessfully?(schema.schemaBody) then
          ActivateTheseSchemata(schema.finalActivationSet)
          return success
        else return failure
      else DeactivateThisSchema(schema)
          return failure
    else return failure
```

Figure 6.8 Problem Solving in Physics 101

The body of a schema can specify an execution ordering among other schemata. Several control constructs are used. The constructs and their syntax and semantics are as follows:

(AND *statement*₁, *statement*₂, . . . , *statement*_{*n*})

Statements are applied left-to-right as long as *success* is returned. If all of the statements are successful, the result is *success*, otherwise *failure*.

(OR *statement*₁, *statement*₂, . . . , *statement*_{*n*})

Statements are applied left-to-right until *success* is returned. If none of the statements are successful, the result is *failure*, otherwise *success*.

(LOOP *statement*₁, *statement*₂, . . . , *statement*_{*n*})

This is similar to *OR*. Statements are also applied left-to-right until *success* results. If none of the statements are successful, the result is *failure*. Otherwise the schema's two precondition groups are again tested. If they are still satisfied, the statements are re-executed. This continues until some precondition fails or all the statements fail. The final result is *success* if any of the statements, in any of the iterations, ever succeeded.

(ALL *statement*₁, *statement*₂, . . . , *statement*_{*n*})

Statements are applied left-to-right. The result is always *success*.

The remainder of this section presents several of **Physics 101**'s problem-solving schemata. This collection of schemata illustrates the features described above and are representative of the full collection contained in the system.

The first sample schema (table 6.5) is **TransformExpression**. It guides the elimination of obstacles from an expression (it is not included in the schema hierarchy presented above). This schemata loops as long as there are obstacles present in the current expression. At each iteration the schema that chooses formulae to substitute is executed. Formulae are chosen according to the three strategies described in chapter 5. If no substitutions can be performed, the backtrack schema moves the system back to the last recorded choice point. If there is none, the fail schema is executed, which requests a solution from the system's teacher.

The second sample schema (table 6.6) attempts to cancel variables. If some substitutions can be found that reduce the number of obstacles, these substitutions are made, the cancelling terms are brought together (by **GroupForCancels**), and the resulting mathematical identities are

Table 6.5 The *TransformExpression* Schema

<i>KeepActive? Conditions</i>	ObstaclesPresent?(expression)
<i>Other Preconditions</i>	true
<i>Initial Activation Set</i>	{}
<i>Schema Body</i>	(LOOP (OR SubstFormulae BackTrack Fail))
<i>Final Activation Set</i>	{}

Table 6.6 The *SubstToCancel* Schema

<i>KeepActive? Conditions</i>	true
<i>Other Preconditions</i>	CanCancelObstacles?(expression)
<i>Initial Activation Set</i>	{SubstEquations, GroupForCancels, SubstIdentities}
<i>Schema Body</i>	(ALL SubstEquations GroupForCancels SubstIdentities)
<i>Final Activation Set</i>	{}

eliminated. The schema *SubstEquations* updates the current expression and, depending on the type of equation substituted, activates other schemata. For example, if a new equation contains integrals or derivatives, the schemata related to solving or rearranging calculus are activated. All new equations lead to the activation of schemata related to rearranging and cancelling algebra, plus the schemata for performing arithmetic. Finally, *SubstEquations* always calls the schema that simplifies the current expression by performing any arithmetic that can be done. The schema called to do this is illustrated next.

Table 6.7 contains the schema for performing any arithmetic that can be done in the current expression. If there are no numbers in the current expression, this schema deactivates itself, otherwise it loops through its body. Looping continues as long as there are numbers in the current expression and one of the sub-schemata can simplify the current expression. Notice that this schema stays active whenever there are numbers in the current expression, even when no further arithmetic can be performed. Some other schema may alter the expression so that additional arithmetic can be performed.

Table 6.7 The *SolveArithmetic* Schema

<i>Keep Active? Conditions</i>	ContainsNumbers?(expression)
<i>Other Preconditions</i>	true
<i>Initial Activation Set</i>	{}
<i>Schema Body</i>	(LOOP (OR MultiplyNumbers AddNumbers SubstIdentities MultiplyByZero))
<i>Final Activation Set</i>	{}

The last sample schema is in table 6.8. This schema checks if there is a zero in the current expression that is multiplying other terms. The multiplication is performed in order to cancel these terms. This schema's body directly alters the current expression, producing a new one, rather than transferring control to other schemata as the above examples did. After cancelling terms, two other schemata are activated. Since the new expression may contain a sub-expression of the form $x + 0$, *SubstAddIdentities* is activated. Similarly, the new expression may contain $<number> + 0$, so *AddNumbers* is activated. In the next iteration of *SolveArithmetic* these schemata would further simplify the expression.

Table 6.8 The *MultiplyByZero* Schema

<i>KeepActive? Conditions</i>	ContainsMultiplyingZeroes?(expression)
<i>Other Preconditions</i>	true
<i>Initial Activation Set</i>	{}
<i>Schema Body</i>	<execute substitutions>
<i>Final Activation Set</i>	{SubstAddIdentities, AddNumbers}

6.4. Summary

This chapter presents additional details about the algorithms in **Physics 101**. The contents of cancellation graphs are explained and their construction is described. Cancellation graphs record how a calculation produces an expression containing only acceptable variables. Attention is focussed on the cancellation of unacceptable variables. These cancellations provide the explanation of why variables other than the unknown appear in the left-hand side of a calculation.

Once the cancellation graph is constructed, it is used to guide the construction of the general version of the calculation, from which a new equation schema is produced. Variables considered unacceptable in the specific calculation are considered unacceptable in its general version. In the general calculation, the general versions of the unacceptable variables are eliminated, based on the assumption that the specific calculation is representative of future calculations.

This technique for mapping variable acceptability from the specific example to the general case is only one possibility. Another approach would involve marking variables as being acceptable or unacceptable in general. This marking could be done *a priori* or could be the result of recording statistics over a number of problems. If this were done, some unacceptable variables in the specific calculation could be considered acceptable in the general calculation and thus remain in the final right-hand side. When a variable in a calculation is considered unacceptable in general but is acceptable in the specific example, a general equation would not be produced.

The structure of problem-solving schemata is also described in this chapter. To increase the efficiency of problem solving, only active schemata are considered. Problem-solving schemata can activate and deactivate other schemata, and can deactivate themselves if they determine they are no longer relevant. Several of the schemata used in **Physics 101** are described.

The next chapter completes the presentation of **Physics 101** by discussing several additional examples, which highlight various aspects of the system.

Chapter 7

Additional Physics 101 Examples

Several additional examples of the operation of **Physics 101** are presented in this chapter. The first example, involving the concept of energy conservation, illustrates the effect that a problem-specific equation has on the generalization process. The next demonstrates how the system can use the result of one learning episode in a later learning episode. The final example shows how fortuitous circumstances in the specific example can influence the learning process. The cancellation graphs and the acquired rules for these examples are presented, as is a special case law for energy conservation.

7.1. Learning about Energy Conservation

A second problem (figure 7.1) presented to **Physics 101** involves a brick falling under the influence of gravity. Again, information at two different states is presented. The mass of the brick, its initial velocity, and its height in the two states are provided. Also, the system is told the net force on the brick is $M g X_y(t)$. The goal is to find the brick's velocity in the second state. **Physics 101**'s problem solver cannot solve this problem using its initial collection of physics formulae. The teacher's solution to this problem uses energy conservation, and is shown

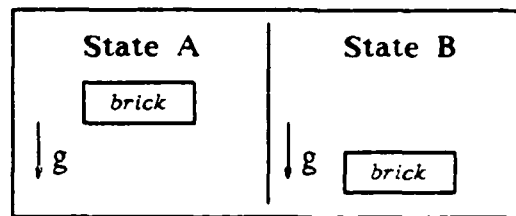


Figure 7.1 A Falling Brick

in figure 7.2. In this solution, the kinetic energy ($\frac{1}{2} \text{mass} \times \text{velocity}^2$) plus the potential energy ($\text{mass} \times g \times \text{height}$) in the two states is equated.

As in the momentum problem, in the provided solution a previously-unseen equation is evaluated at two different points, and the two results equated. In order to justify the use of this equation, the system first attempts to produce an expression, where time is explicit, that

$$\frac{1}{2} M_1 V_{1,y}^2(A) + M_1 g X_{1,y}(A) = \frac{1}{2} M_1 V_{1,y}^2(B) + M_1 g X_{1,y}(B)$$

$$\frac{1}{2} (2 \text{ kg}) (0 \frac{\text{m}}{\text{s}^2}) + (2 \text{ kg}) (9.8 \frac{\text{m}}{\text{s}^2}) (5 \text{ m}) = \frac{1}{2} (2 \text{ kg}) V_{1,y}^2(B) + (2 \text{ kg}) (9.81 \frac{\text{m}}{\text{s}^2}) (0 \text{ m})$$

$$0 \frac{\text{kg m}^2}{\text{s}^2} + 98.1 \frac{\text{kg m}^2}{\text{s}^2} = 1 \text{ kg } V_{1,y}^2(B) + 0 \frac{\text{kg m}^2}{\text{s}^2}$$

$$98.1 \frac{\text{kg m}^2}{\text{s}^2} = 1 \text{ kg } V_{1,y}^2(B)$$

$$V_{1,y}^2(B) = 98.1 \frac{\text{m}^2}{\text{s}^2}$$

$$V_{1,y}(B) = 9.91 \frac{\text{m}}{\text{s}}$$

Figure 7.2 The Teacher's Solution to the Energy Problem

describes the general form of one side of the initial equation in figure 7.2. The system's problem solver cannot meet this goal, so another approach is taken. In the second approach, the system takes the temporal derivative of the underlying expression to see if it is zero. If so, it is valid to equate the expression at any two time points. The second approach succeeds, using the calculation steps shown in figure 7.3.

The cancellation graph for the calculation of figure 7.3 is presented in figure 7.4. There are two members in the obstacle set of the unknown velocity. One is the net force on the falling object, while the other is the velocity itself. This possibly confusing state of affairs, where the

$$\begin{aligned}
 & \frac{d}{dt} \left(\frac{1}{2} M_1 V_{1,y}^2(t) + M_1 g X_{1,y}(t) \right) \\
 1 \quad \text{SeparateDerivatives} &= \frac{d}{dt} \left(\frac{1}{2} M_1 V_{1,y}^2(t) \right) + \frac{d}{dt} (M_1 g X_{1,y}(t)) \\
 2 \quad \text{ConstsOutOfDerivatives} &= \frac{1}{2} M_1 \frac{d}{dt} V_{1,y}^2(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
 3 \quad \text{Differentiate} &= \frac{2}{2} M_1 V_{1,y}(t) \frac{d}{dt} V_{1,y}(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
 4 \quad \text{MultiplyNumbers} &= 1 M_1 V_{1,y}(t) \frac{d}{dt} V_{1,y}(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
 5 \quad \text{RemoveIdentities} &= M_1 V_{1,y}(t) \frac{d}{dt} V_{1,y}(t) + M_1 g \frac{d}{dt} X_{1,y}(t) \\
 6 \quad \text{Subst:Calculus} &= M_1 V_{1,y}(t) A_{1,y}(t) + M_1 g V_{1,y}(t) \\
 7 \quad \text{Subst:ToCancel} &= \frac{M_1 V_{1,y}(t) F_{net,1,y}(t)}{M_1} - \frac{M_1 F_{net,1,y}(t) V_{1,y}(t)}{M_1} \\
 8 \quad \text{Subst:MultiIdentities} &= 1 V_{1,y}(t) F_{net,1,y}(t) - 1 F_{net,1,y}(t) V_{1,y}(t) \\
 9 \quad \text{RemoveIdentities} &= V_{1,y}(t) F_{net,1,y}(t) - F_{net,1,y}(t) V_{1,y}(t) \\
 10 \quad \text{Subst:AddIdentities} &= 0 \frac{kg \cdot m^2}{s^3}
 \end{aligned}$$

Figure 7.3 Verifying the Teacher's Solution to the Energy Problem

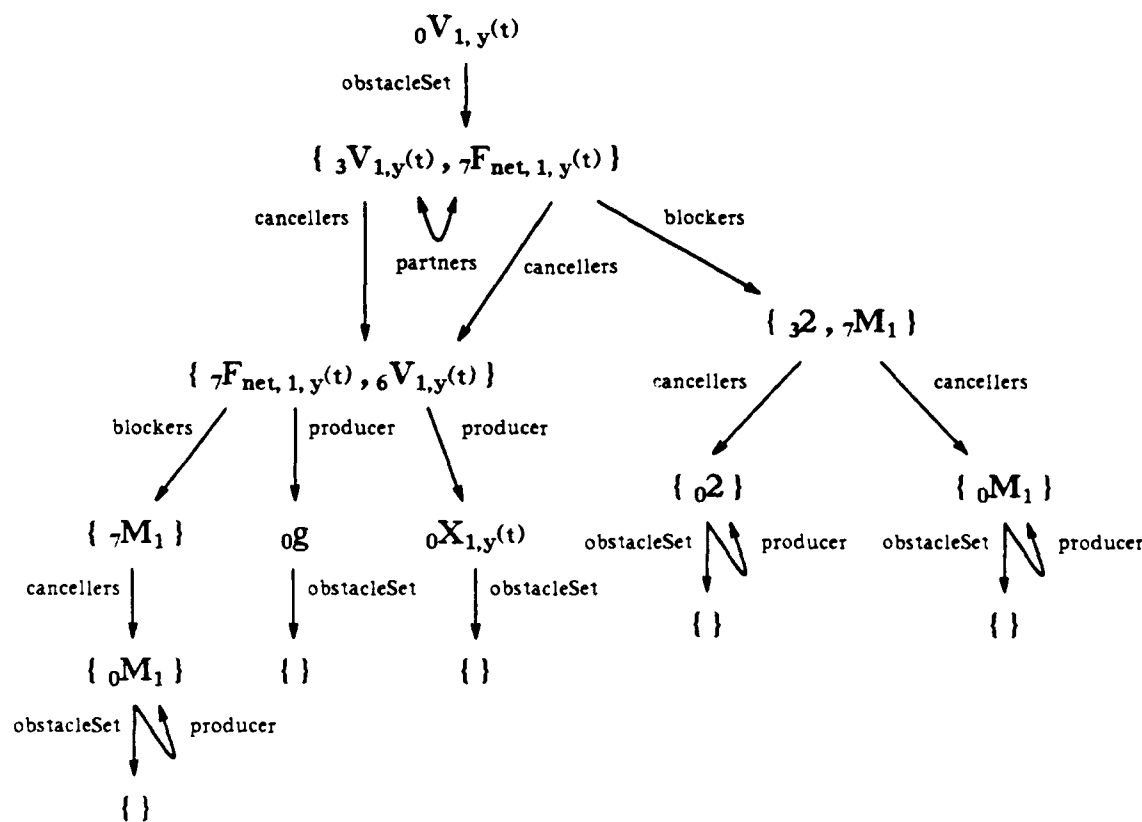


Figure 7.4 The Cancellation Graph for the Energy Problem

obstacle set contains the unknown *plus* another variable. arises because the calculus rule

$$\frac{d}{dt} ?x^{?n} = ?n ?x \frac{d}{dt} ?x^{?n-1}$$

is used in line 3 of figure 7.3. This rule replaces the original instantiation of $V_{1,y}$ with two new, distinct instantiations. The two primary obstacles are partners in the cancellation of line 10. However, before this additive cancellation can take place, the multiplicative blockers of $F_{net,1,y}$ must be eliminated. There are two blockers: the mass term introduced in line 7, and the number 2 introduced in line 3. (Although not shown in figure 7.4, the 2 is also a blocker of $V_{1,y}$.) Both of these blockers are cancelled by terms that are present in the left-hand side of the calculation, and their cancellation introduces no new obstacles.

The cancellers of the primary obstacles are produced by the remaining variables in the left-hand side of the calculation. Since M_1 is blocking the net force, the producer of its

canceller must also appear in the left-hand side. Requiring the presence of the variables that produce the cancellers does not lead to any more obstacles. The role of each term in the initial expression is understood.

The next several figures illustrate the construction of the general calculation. In the first figure, figure 7.5, the primary obstacles are derived from the generalized unknown.

Next, the blockers of the primary obstacles must be eliminated. Figure 7.6 shows how these unblockers are introduced into the calculation.

Finally, the cancellers of the primary obstacles are introduced into the calculation, as shown in figure 7.7. Because the formula $F_{net} = MgX$ is marked as not holding in all worlds, it is not used in the general calculation. Instead, the calculation is reconstructed without using this formula, which means that the net force needed to cancel the primary obstacles must appear in the left-hand side of the calculation. These new left-hand side variables produce no secondary obstacles, so reconstruction of the calculation is complete.

The general law **Physics 101's** produces by analyzing the sample solution is presented in figure 7.8.

This general energy conservation law (figure 7.8) applies whenever the total force on an object is known. Notice, though, that a potentially complicated integral needs to be computed if this general law is to be used. To use this formula, it is not sufficient to possess knowledge of the values of variables at two different times. A problem solver must also know how the net force depends on position for a continuum of times. In the specific problem there is a *constant* net force (gravity), and when this force is constant the problem is greatly simplified. Integrating a constant force leads to a potential energy determined by that constant force multiplied by the object's position. The position only needs to be known at the two distinct times, and not for *all* intervening times. The special case schema for energy conservation is contained in figure 7.9. Again, since this is a conservation schema, the time at which each state occurs need not be known.

7.2. Learning about the Sum of Internal Forces

In a third exercise, **Physics 101** is again given a problem describing the state of a collection of three balls at two different times. The internal forces of balls 2 and 3 are specified for both

$$\begin{aligned}
& \frac{d}{dt} V^2_{\gamma_s, \gamma_c}(t) \\
(2) \quad &= 2 V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) \\
(5) \quad &= 2 V_{\gamma_s, \gamma_c}(t) A_{\gamma_s, \gamma_c}(t) \\
(6) \quad &= 2 V_{\gamma_s, \gamma_c}(t) \frac{F_{\gamma_s, \gamma_c}(t)}{M_{\gamma_s}}
\end{aligned}$$

Figure 7.5 Introducing the Primary Obstacles

$$\begin{aligned}
& \frac{d}{dt} \left(\frac{1}{2} M_{\gamma_c} V^2_{\gamma_s, \gamma_c}(t) \right) \\
(1) \quad &= \frac{1}{2} M_{\gamma_c} \frac{d}{dt} V^2_{\gamma_s, \gamma_c}(t) \\
(2) \quad &= \frac{2}{2} M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) \\
(3) \quad &= 1 M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) \\
(4) \quad &= M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) \\
(5) \quad &= M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) A_{\gamma_s, \gamma_c}(t) \\
(6) \quad &= \frac{M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) F_{\gamma_s, \gamma_c}(t)}{M_{\gamma_s}} \\
(7) \quad &= 1 V_{\gamma_s, \gamma_c}(t) F_{\gamma_s, \gamma_c} \\
(8) \quad &= V_{\gamma_s, \gamma_c}(t) F_{\gamma_s, \gamma_c}
\end{aligned}$$

Figure 7.6 Introducing Terms to Unblock the Primary Obstacles

$$\begin{aligned}
& \frac{d}{dt} \left(\frac{1}{2} M_{\gamma_c} V_{\gamma_s, \gamma_c}^2(t) - \int F_{net, \gamma_s, \gamma_c}(t) dX_{\gamma_s, \gamma_c}(t) \right) \\
(1) \quad &= \frac{1}{2} M_{\gamma_c} \frac{d}{dt} V_{\gamma_s, \gamma_c}^2(t) - \frac{d}{dt} \int F_{net, \gamma_s, \gamma_c}(t) dX_{\gamma_s, \gamma_c}(t) \\
(2) \quad &= \frac{2}{2} M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) - F_{net, \gamma_s, \gamma_c}(t) \frac{d}{dt} X_{\gamma_s, \gamma_c}(t) \\
(3) \quad &= 1 M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) - F_{net, \gamma_s, \gamma_c}(t) \frac{d}{dt} X_{\gamma_s, \gamma_c}(t) \\
(4) \quad &= M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) \frac{d}{dt} V_{\gamma_s, \gamma_c}(t) - F_{net, \gamma_s, \gamma_c}(t) \frac{d}{dt} X_{\gamma_s, \gamma_c}(t) \\
(5) \quad &= M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) A_{\gamma_s, \gamma_c}(t) - F_{net, \gamma_s, \gamma_c}(t) V_{\gamma_s, \gamma_c}(t) \\
(6) \quad &= \frac{M_{\gamma_c} V_{\gamma_s, \gamma_c}(t) F_{\gamma_s, \gamma_c}(t)}{M_{\gamma_s}} - F_{net, \gamma_s, \gamma_c}(t) V_{\gamma_s, \gamma_c}(t) \\
(7) \quad &= 1 V_{\gamma_s, \gamma_c}(t) F_{\gamma_s, \gamma_c} - F_{net, \gamma_s, \gamma_c}(t) V_{\gamma_s, \gamma_c}(t) \\
(8) \quad &= V_{\gamma_s, \gamma_c}(t) F_{\gamma_s, \gamma_c} - F_{net, \gamma_s, \gamma_c}(t) V_{\gamma_s, \gamma_c}(t) \\
(9) \quad &= 0 \frac{kg \cdot m^2}{s^3}
\end{aligned}$$

Figure 7.7 Introducing the Cancellers of the Primary Obstacles

Equation

$$\frac{d}{dt} \left[\frac{1}{2} M_{\gamma_i} V_{\gamma_i, \gamma_c}^2(t) - \int F_{net, \gamma_i, \gamma_c}(t) dX_{\gamma_i, \gamma_c} \right] = 0 \frac{kg \cdot m^2}{s^3}$$

Preconditions

Object(?i) \wedge
 IsaComponent(?c) \wedge
 IndependentOf(M_{γ_i}, t) \wedge
 NOT(ZeroValued(M_{γ_i}))

Figure 7.8 The Final Result of the Energy Problem

Equation

$$\begin{aligned} & \frac{1}{2} M_{\gamma_i} V^2_{\gamma_i} (?t_1) + M_{\gamma_i} g X_{\gamma_i, \gamma_c} (?t_1) \\ &= \frac{1}{2} M_{\gamma_i} V^2_{\gamma_i} (?t_2) + M_{\gamma_i} g X_{\gamma_i, \gamma_c} (?t_2) \end{aligned}$$

Preconditions

Object(?i) \wedge
IsaComponent(?c) \wedge
IndependentOf(M_{γ_i}, t) \wedge
NOT(ZeroValued(M_{γ_i})) \wedge
SpecificPointOf(?t₁, t) \wedge
SpecificPointOf(?t₂, t) \wedge
?t₁ \neq ?t₂

Eliminated Terms

?t₁, ?t₂

Special Case Conditions

$$F_{net, \gamma_i}(t) = M_{\gamma_i} g \gamma_c$$

Figure 7.9 The Special-Case Energy Law

states, while ball 1's internal force is only known in the initial state. The goal is to determine the internal force on ball 1 in the second state. In the solution provided by the teacher, the sum of the three internal forces is evaluated in each state, and the results equated. This new equation must be explained. As shown in figure 7.10, the system is able to construct an expression describing how the sum of the internal forces depends on time. Since it is constant, it can be equated in any two states.

Figure 7.11 contains the cancellation graph for the calculation in figure 7.9. This problem is actually a sub-problem of the momentum example, and the two problems' cancellation graphs are somewhat similar. The main difference is that in the internal force problem there are no cancellation blockers. As in the momentum example, the primary obstacle set contains the two

$$F_{int,1,z}(t) + F_{int,2,z}(t) + F_{int,3,z}(t)$$

- 1 SubstSameType $= F_{2,1,z}(t) + F_{3,1,z}(t) + F_{1,2,z}(t) + F_{3,2,z}(t) + F_{1,3,z}(t) + F_{2,3,z}(t)$
- 2 SubstToCancel $= F_{2,1,z}(t) + F_{3,1,z}(t) - F_{2,1,z}(t) + F_{3,2,z}(t) - F_{3,1,z}(t) - F_{3,2,z}(t)$
- 3 SubstAddIdentities $= 0 \frac{kg \ m}{s^2} + 0 \frac{kg \ m}{s^2} + 0 \frac{kg \ m}{s^2}$
- 4 AddNumbers $= 0 \frac{kg \ m}{s^2}$

Figure 7.10 Verifying the Teacher's Solution to the Internal Force Problem

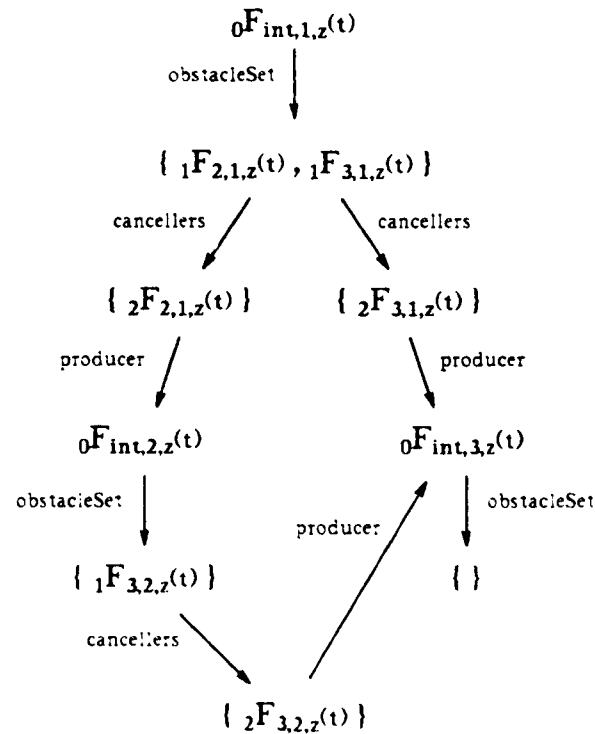


Figure 7.11 The Cancellation Graph for the Internal Force Problem

inter-object forces acting on ball 1. These unblocked obstacles are cancelled by forces descended from the internal forces of the other two balls. However, cancelling the primary obstacles produces a secondary obstacle — the force between balls 2 and 3. Cancelling this obstacle does not produce any additional obstacles.

The result of generalizing the internal force calculation is presented in figure 7.12. Again, *all* the inter-object forces acting on an object must be cancelled. This results in the need for the presence of every object's internal force in the left-hand side of the equation. With this left-hand side, all the inter-object forces cancel, and **Physics 101** acquires the concept that the sum of the internal forces of any system is conserved.

7.3. Using the New Force Law to Learn about Momentum

This example demonstrates the important property that **Physics 101** can use, in subsequent problem solving and learning episodes, an equation schema it acquires. The initial momentum problem (figure 3.1) is rerun after the system acquires the concept illustrated in figure 7.12. Figure 7.13 contains those lines in figure 3.5 that change. (Line 7 of figure 3.5 has no corresponding step in this calculation. That is, the system goes directly from line 6' to line 8'.) Now that **Physics 101** has an equation specifying a useful relationship among the internal forces, it can be used to eliminate the three internal forces from the calculation. There is no need to rewrite them in terms of the inter-object forces.

The cancellation graph for this calculation is in figure 7.14. This graph is substantially different from the one for the original solution to the momentum problem (figure 3.8). In this problem the only primary obstacle is $F_{int,1,2}$, which is blocked by a mass term and cancelled by an internal force descended from the velocity of ball 3. Cancelling the primary obstacle introduces one secondary obstacle, namely $F_{int,2,3}$. In order to cancel this obstacle, $V_{2,3}$ must

Equation

$$\sum_{j \in ObjectsInWorld} F_{int,j,c}(?t) = 0 \frac{kg \cdot m}{s^2}$$

Preconditions

$IsaTime(?t) \wedge IsaComponent(?c)$

Figure 7.12 The Final Result of the Internal Force Problem

$$\begin{aligned}
6' \quad \text{SubstSameType} &= \int (F_{ext,1,x}(t) + F_{int,1,x}(t)) dt + \int (F_{ext,2,x}(t) + F_{int,2,x}(t)) dt \\
&\quad + \int (F_{ext,3,x}(t) + F_{int,3,x}(t)) dt \\
8' \quad \text{SubstToCancel} &= \int (F_{ext,1,x}(t) + F_{int,1,x}(t)) dt + \int (F_{ext,2,x}(t) + F_{int,2,x}(t)) dt \\
&\quad + \int (F_{ext,3,x}(t) - F_{int,1,x}(t) - F_{int,2,x}(t)) dt \\
9' \quad \text{CombineCalculus} &= \int (F_{ext,1,x}(t) + F_{int,1,x}(t) + F_{ext,2,x}(t) + F_{int,2,x}(t) \\
&\quad + F_{ext,3,x}(t) - F_{int,1,x}(t) - F_{int,2,x}(t)) dt \\
10' \quad \text{SubAddIdentities} &= \int (F_{ext,1,x}(t) + 0 \frac{kg \cdot m}{s^2} + F_{ext,2,x}(t) + 0 \frac{kg \cdot m}{s^2} + F_{ext,3,x}(t)) dt
\end{aligned}$$

Figure 7.13 Verifying the Momentum Problem after the Internal Force Problem

be present in the left-hand of the calculation. The presence of this variable, and the variable needed to cancel its blocker, does not introduce any additional obstacles. Even though their cancellation graphs differ, generalizing this calculation produces the same result as generalizing the calculation in figure 3.5.

7.4. Attempting to Learn from a Two-Ball Collision

It was stated earlier that at least a three-ball collision problem is needed to properly acquire the concept of momentum conservation. This section describes the results of analyzing a two-ball collision. The two-ball problem is identical to the original three-ball problem of figure 3.1, except that ball 3 is not included. Again, the teacher equates the momentum in the two states and the system needs to verify this new equation. Verification is nearly identical to the calculation in figure 3.5 and is not shown here. The main difference is that variables involving ball 3 are not present.

The cancellation graph for this simpler calculation is contained in figure 7.15. It is virtually identical to the left half of figure 3.8. The significant difference is that the presence of $V_{2,x}$ produces no obstacles because cancelling the obstacles of the unknown cancels all the potential obstacles of this variable. The fortuitous circumstances of the two-body example mean that the problem of what to do with the additional inter-object forces need not be faced. This difference becomes important in the generalization phase, as described in section 4.1. Since

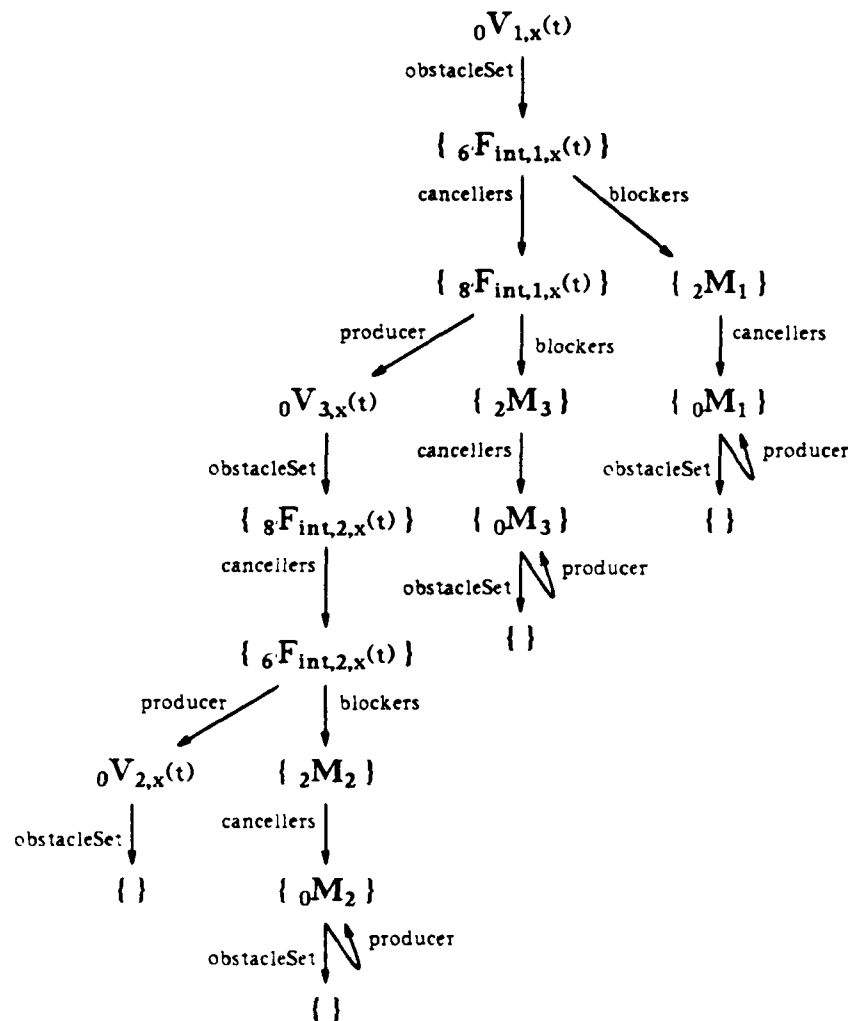


Figure 7.14 The Cancellation Graph for the Second Solution of the Momentum Problem

Physics 101 does no problem solving during generalization, these inter-object forces remain, as shown in figure 7.16.

One way to handle the case where there are no specific representatives of a variable in the general calculation would be to save the final result, marking it for possible later refinement. However, in **Physics 101** whenever there are variables without representatives in the specific calculation, no learning takes place.

This example illustrates an important issue in generalizing the structure of explanations. When generalizing an explanation to handle more entities, a learning system must be able to

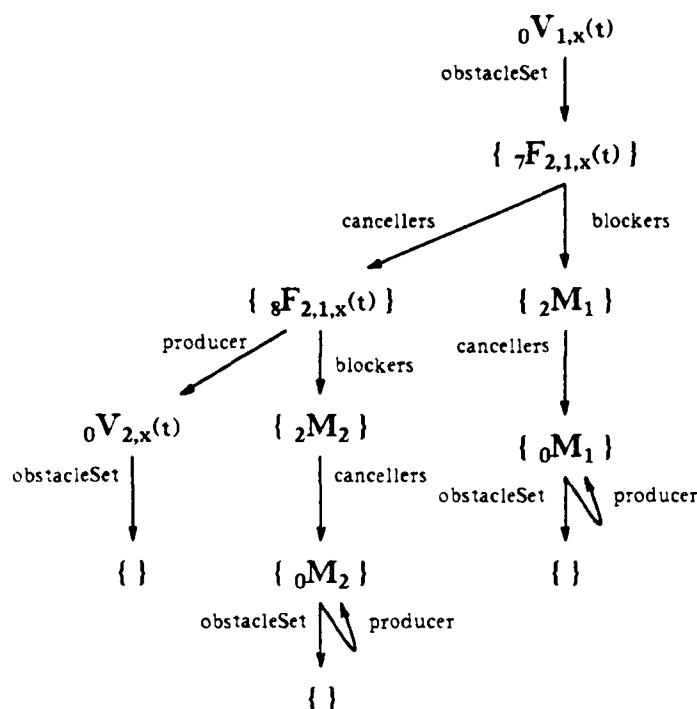


Figure 7.15 The Cancellation Graph for a Two-Ball Momentum Problem

recognize any new issues that arise due to the presence of these additional entities. The sample problem's solution may not have had to address all of these issues, due to fortuitous circumstances in the example, and the learner must possess some facility to recognize and surmount these extra issues.

7.5. Summary

This chapter presents several additional examples of the operation of **Physics 101**. An example involving energy conservation is presented first. This example demonstrates **Physics 101** acquiring a second fundamental concept of physics. It also illustrates how the system generalizes the structure of explanations in a way different than generalizing number. The energy example used a equation schema specific to that problem, and in the generalized calculation that equation is not used. However, a special-case is produced in which the limited rule is used.

Equation

$$\frac{d}{dt} \sum_{i \in \text{ObjectsInWorld}} M_i V_{i, ?c}(t) = \sum_{i \in \text{ObjectsInWorld}} F_{\text{ext}, i, ?c}(t) + \sum_{j \neq ?s} \sum_{k \neq j, ?s} F_{k, j, ?c}(t)$$

Preconditions

IsaComponent(?c) \wedge
Member(?s, ObjectsInWorld) \wedge
 $\forall i \in \text{ObjectsInWorld}$ NOT(ZeroValued(M_i)) \wedge
 $\forall i \in \text{ObjectsInWorld}$ IndependentOf(M_i, t)

Eliminated Terms

$$\forall i \neq ?s \quad F_{?s, i, ?c}(t) \quad , \quad F_{i, ?s, ?c}(t)$$

Figure 7.16 The Potential Final Result of the Two-Ball Momentum Problem

In the second example of this chapter, **Physics 101** learns that the sum of a world's internal forces is always zero. The original momentum example of chapter 3 is rerun after the system learns the internal force law. Assistance from its teacher is still needed, but **Physics 101** is able to use the internal force law when learning the momentum law. This demonstrates the system's ability to use the results of its learning to facilitate additional knowledge acquisition.

Finally, a two-body collision example is presented. The system does not acquire enough information from this example to properly acquire the momentum law. Some lurking obstacles that appear in the general case need not be faced to solve this simple problem.

Chapter 8

Overview of the BAGGER System

As is stated in the introduction, most research in explanation-based learning involves relaxing constraints on the variables in the explanation of a specific example, rather than generalizing the structure of the explanation itself. However, this precludes the acquisition of concepts in which an iterative process is implicitly represented in the explanation by a fixed number of applications. Such explanations must be augmented during generalization. This can involve generalizing such things as the number of entities involved in a concept or the number of times some action is performed.

In **Physics 101**, the need for generalizing the structure of an explanation is motivated by an analytic justification of an example's solution and general domain knowledge. In the momentum problem, information localized in a single physics formula leads to a global restructuring of a specific solution's explanation. **Physics 101** is designed to reason about the use of mathematical formulae. Its generalization algorithm takes great advantage of the properties of algebraic cancellation and is based on the assumption that the general version of these cancellations should be represented in the general concept acquired.

However, as previously discussed, the need to generalize the structure of explanations is ubiquitous. It occurs in many domains besides those that involve mathematics. This chapter and the next several present a second approach to the problem of generalizing explanation structures. In this approach, the *form* of the explanation motivates generalizing its structure. The explanation-based generalization algorithm developed is domain-independent — no special characteristics of any particular domain are used as the foundation of the algorithm.

The **BAGGER** system (Building Augmented Generalizations by Generating Extended Recurrences) attempts to construct concepts that involve generalizing to N . The fully-implemented **BAGGER** system analyzes explanation structures and detects extendible repeated, inter-dependent applications of rules. When any are found, the explanation is extended so that an arbitrary number of repeated applications of the original rule are supported. The final structure is then generalized and a new rule produced, which embodies a crucial shift in representation.

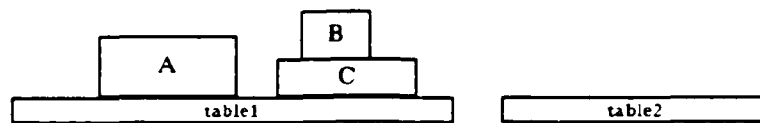
In this chapter, several examples requiring generalization to N are introduced. Most of the examples investigated use the situation calculus [McCarthy63] to reason about actions, in the style of [Green69]. (Green's formulation is also discussed in [Nilsson80].) A description of situation calculus follows the presentation of the examples. The following two sections describe the type of rules acquired by **BAGGER**. Later chapters elaborate the **BAGGER** algorithm and demonstrate the performance gains achieved by it.

8.1. Some Sample Problems

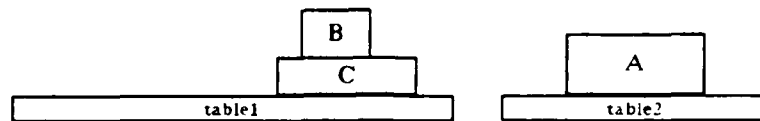
Several examples are presented in this section to illustrate the operation of **BAGGER**.

One problem solution analyzed by **BAGGER** is shown in figure 8.1. The goal is to place a properly-supported block so that its center is above the dotted line and within the horizontal confines of the line. **BAGGER** is provided low-level domain knowledge about blocks, including how to transfer a single block from one location to another and how to calculate its new horizontal and vertical position. Briefly, to move a block it must have nothing on it and there must be free space at which to place it. The system produces a situation calculus proof validating the actions shown in figure 8.1, in which three blocks must be moved to build the tower.

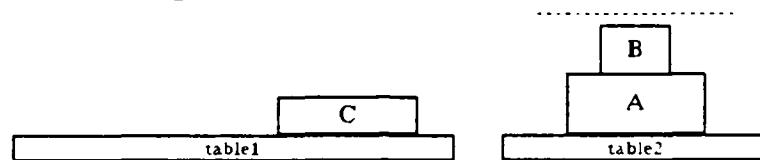
Situation₀



Situation₁



Situation₂



Situation₃



Figure 8.1 Constructing a Three-Block Tower

If a standard explanation-based generalization algorithm is applied to the resulting proof, a plan for moving *three* blocks will result. They need not be these same three blocks — any three distinct ones will suffice. Nor is it necessary that the first block moved be placed on a table, any flat, clear surface is acceptable. Finally, the height of the tower need not be the same as that in the specific example. Given appropriately sized blocks, towers of any height can be constructed. Many characteristics of the problem are generalized. However, the fact that exactly three blocks are moved would remain.

If one considers the universe of all possible towers, as shown in figure 8.2, only a small fraction of them would be captured by the acquired rule. Separate rules would need to be learned for towers containing two blocks, five blocks, etc. What is desired is the acquisition of a rule that describes how towers containing any number of blocks can be constructed.

By analyzing the proof of the construction of the three-block tower, **BAGGER** acquires a general plan for building towers by stacking *arbitrary* numbers of blocks, as illustrated in figure 8.3. This new plan incorporates an indefinite number of applications of the previously known plan for moving a single block.

In another example, the system observes three blocks being removed from a stack in order to satisfy the goal of having a specific block be clear. Extending the explanation of these actions

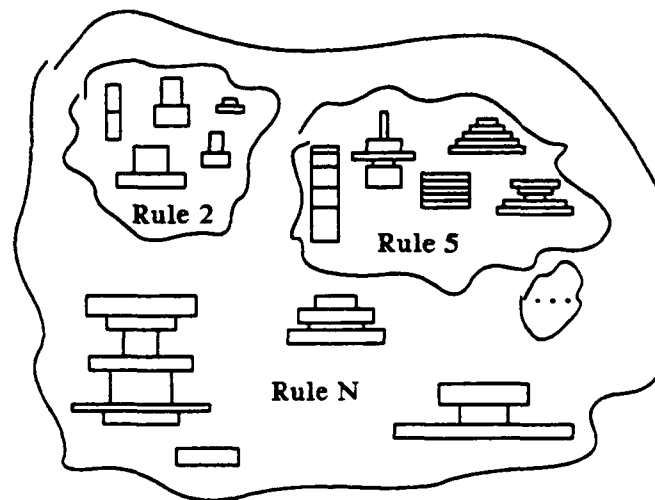


Figure 8.2 Universes of Constructible Towers

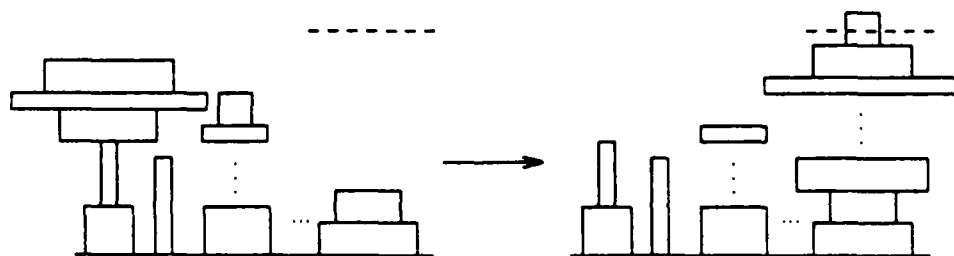


Figure 8.3 A General Plan for Constructing Towers

produces a plan for unstacking any number of blocks in order to clear a block within the stack. Figure 8.4 illustrates this general plan. The plan includes the system's realization that the last unstacked block is currently clear and thus makes a suitable site on which to place the next block to be moved. This knowledge is incorporated into the plan and no problem solving need be performed finding destinations once the first free location is found.

Unlike many other block-manipulation examples, in these examples it is *not* assumed that blocks can support only one other block. This means that moving a block does not necessarily clear its supporting block. Another concept learned by BAGGER, by observing two blocks being moved from on top a table, is a general plan for clearing an object directly supporting any number of clear blocks. This plan is illustrated in figure 8.5.

The domain of digital circuit design also contains many examples where a fixed number of rule applications should be generalized into an arbitrary number. By observing the repeated application of DeMorgan's law to implement two cascaded *AND* gates using *OR* and *NOT* gates, BAGGER produces a general version of DeMorgan's law which can be used to implement N cascaded *AND* gates with N *OR* and one *NOT* gate. This example, which does not use situation calculus, is shown in figure 8.6.

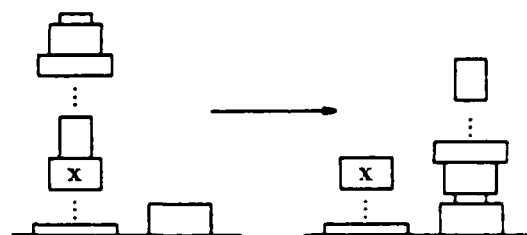


Figure 8.4 A General Plan for Unstacking Towers

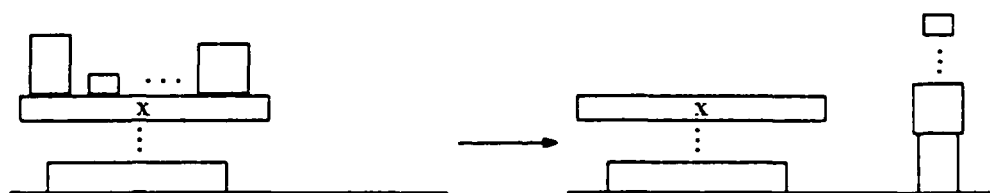


Figure 8.5 A General Plan for Clearing Objects

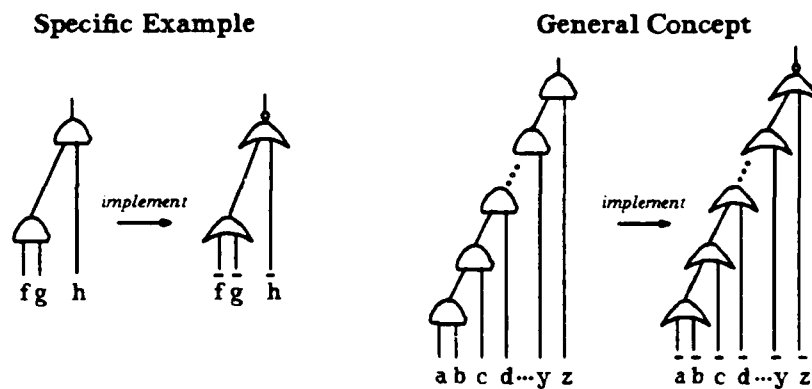


Figure 8.6 A Circuit Design Example

8.2. Situation Calculus

In situation calculus, predicates and functions whose values may change over time are given an extra argument which indicates the situation in which they are being evaluated. For example, rather than using the predicate $On(x,y)$, indicating that x is on y , the predicate $On(x,y,s)$ is used, indicating that in situation s , x is on y . In this formulation, operators are represented as functions that map from one situation to another.

Problem solving with **BAGGER**'s situation calculus rules can be viewed as transforming and expanding situations until one is found in which the goal is known to be achieved. The **BAGGER** system has two types of inference rules: *inter-situational* rules which specify attributes that a new situation will have after application of a particular operator, and *intra-situational* rules which can embellish **BAGGER**'s knowledge of a situation by specifying additional conclusions that can be drawn within that situation.

Each inter-situational inference rule specifies knowledge about one particular operator. However, operators are not represented by exactly one inference rule. A major inference rule specifies most of the relevant problem-solving information about an operator. But it is augmented by many lesser inference rules which capture the operator's frame axioms and other facts about a new situation. This paradigm contrasts with the standard **STRIPS** [Fikes71] formalism.¹ The inference rules of a **STRIPS**-like system are in a one-to-one correspondence with

¹ Fahlman [Fahlman74] and Fikes [Fikes75] augmented the standard **STRIPS** model by allowing a distinction between primary and secondary relationships. Primary relationships are asserted directly by

the system's operators. Each inference rule fully specifies an operator's add- and delete-lists. These lists provide all of the changes needed to transform the current situation into the new situation. Any state not mentioned in an add- or delete-list is assumed to persist across the operator's application. Thus, the new situation is completely determined by the inference rule. In the **BAGGER** system this is not the case. Many separate inference rules are used to fully characterize the effect of an operator.

The advantage of the **STRIPS** approach is that the system can always be assured that it has represented all that there is to know about a new situation. However, this can also be a disadvantage. A **STRIPS**-like system must always muddle through all there is to know about a situation, no matter how irrelevant many facts may be to the current problem. Conversely, the advantages of **BAGGER**'s approach are that the inference rules are far less complex and therefore more manageable, the system's attention focussing is easier because it does not bog down in situations made overly-complex by many irrelevant facts, and a programmer can more easily write and update knowledge about operators. Furthermore, **STRIPS**-style operators do not allow disjunctive or conditional effects in their add- or delete-lists.

A potential disadvantage of **BAGGER**'s approach is that to completely represent the effects of applying an operator in a particular situation, the system must retrieve all of the relevant inference rules. However, this is not a task that arises in **BAGGER**'s problem solving. Indeed, there has been no attempt to guarantee the completeness of the system's inferential abilities. This means that there may be characteristics of a situation which **BAGGER** can represent but cannot itself infer.

8.3. Sequential Rules

Like its standard inference rules, number-generalized rules in the **BAGGER** system are usually represented in situational calculus. In the previous section, two types of **BAGGER** inference rules are discussed: intra- and inter-situational rules. To define number-generalized rules, the inter-situational rules are further divided into two categories: simple inter-situational rules and *sequential* inter-situational rules (or simply *sequential rules*). Sequential rules apply a

operators while secondary relationships are deduced from the primary ones as needed. While this serves the same purpose as **BAGGER**'s intra-situational rules, multiple inter-situational rules for an operator are not allowed [Waldinger77].

variable number of operators. Thus, within each application of a sequential rule many intermediate situations may be generated. The actual number of intermediate situations depends on the complexity of the problem to be solved. The rule for building towers is an example of a sequential rule. This rule is able to construct towers of any number of blocks in order to achieve a specified goal height (the dotted line). The rule itself decides how many blocks are to be used and selects which blocks to use from among those present in the current situation.

Sequential rules, like their simple inter-situational counterparts, have an antecedent and a consequent. Also, like the simple versions, if the antecedent is satisfied, the consequent specifies properties of the resulting situation. Unlike the simple rules, the resulting situation can be separated from the initial situation by many operator applications and intermediate situations. For example, to build a tower, many block-moving operations must be performed. It is an important feature of sequential rules that no planning need be done in applying the intermediate operators. That is, if the antecedent of a sequential rule is satisfied, its entire sequence of operators can be applied without the need for individually testing or planning for the preconditions. The preconditions of each operator are guaranteed to be true by the construction of the sequential rule itself. Thus, the consequent of a sequential rule can immediately assert properties which must be true in the final situation. A sequential rule behaves much as a STRIPS-like macro-operator. It is termed a *sequential rule* and not a *macro-operator* because it is, in fact, a situational calculus rule and not an operator. It has a situation variable, does not specify *ADD* and *DELETE* lists, etc.

Sequential rules can be much more efficient than simply chaining together simple constituents. This improved efficiency is largely derived from three sources: 1) collecting together antecedents so that redundant and subsumed operator preconditions are eliminated, 2) heuristically ordering the antecedents, and, especially, 3) eliminating antecedents that test operator preconditions which, due to the structure of the rule, are known to be satisfied.

8.4. Representing Sequential Knowledge

A representational shift is crucial to BAGGER's solution to the generalization to N problem. While objects in a world are represented within simple inference rules directly as predicate calculus variables, this is not possible for BAGGER's sequential rules. A standard operator

interacts with a known number of objects. Usually, this number is small. The rule representing the operator that moves blocks, for example, might take as arguments the block to be moved and the new location where it is to be placed. A simple inter-situational rule for this operator might specify that in the resulting situation, the block represented by the first argument is at the location specified by the second. This rule represents exactly one application of the move operator. There are always two arguments. They can be conveniently represented by predicate calculus variables. That is, each of the world objects with which a simple operator interacts can be uniquely named with a predicate calculus variable.

Sequential rules cannot uniquely name each of the important world objects. A rule for building towers must be capable of including an arbitrary number of blocks. The uninstantiated rule cannot know whether it is to be applied next to build a tower of five, seven, or 24 blocks. Since the individual blocks can no longer be named by unique variables within the rule, a shift is necessary to a scheme that can represent aggregations of world objects. Such a representational shift, similar to [Weld86], makes explicit attributes that are only implicitly present in the example. Thus, it shares many characteristics of constructive induction [Michalski83, Rendell85, Watanabe87].

A new object called an *RIS* (for Rule Instantiation Sequence) is introduced to represent arbitrarily large aggregations of world objects. A sequential rule works directly with one of these generalized structures so that it need not individually name every world object with which it interacts. A sequential rule's *RIS* is constructed in the course of satisfying its antecedent. Once this is done, the *RIS* embodies all of the constraints required for the successive application of the sequence of operators that make up the plan.

8.5. Summary

This chapter introduces another approach to generalizing the structure of explanations. The fully-implemented **BAGGER** system analyzes explanation structures expressed in predicate calculus and detects extendible repeated, inter-dependent applications of rules. When any are found, the explanation is extended so that an arbitrary number of repeated applications of the original rule are supported. The final structure is then generalized and a new rule, which embodies a crucial shift in representation, is produced. The **BAGGER** system is capable of

acquiring concepts where an iterative process is only implicitly represented in a specific problem's explanation.

The rules learned by the system allow a problem solver to perform substantially better in future problems. The efficiency gains of **BAGGER** occur for several reasons. One, as is usual in explanation-based learning, the interconnection of a large number of rules is compiled into the learned rule, saving a problem solver the trouble of having to repeatedly search its collection of rules and re-connecting the same rules. This repeated effort can be further slowed down if the problem solver follows many unfruitful pathways. Two, the antecedents of a new sequential rule are heuristically ordered to increase the efficiency of satisfying them. Three, preconditions that, due to the structure of the rule, are known to be satisfied do not appear in the final rule and, hence, no time need be wasted testing them. Four, the preconditions for sequential rules are expressed only in terms of the initial situation. This allows a problem solver to jump from the current situation to some other situation arbitrarily many states away without reasoning about any of the intermediate states. These gains are even more significant in **BAGGER** because its sequential rule representation incorporates an indefinite number of ordinary rules.

Situation calculus provides an appealing way of representing changing environments. While it generalizes examples formulated in standard predicate calculus, **BAGGER** is particularly well-suited for problems expressed in situation calculus. Portions of its generalization algorithm take advantage of the frame axioms and related inter-situation rules used in situation calculus.

Unlike standard predicate calculus rules, a sequential rule cannot always use predicate calculus variables to represent individual objects in a world. Since a sequential rule must be able to accommodate any number of objects, a more abstract representation is needed. A data structure called a *rule instantiation sequence* (RIS) is used to support this task.

The next chapter presents the **BAGGER** generalization algorithm. Following that, there is a detailed presentation of the above examples, including their full proof trees and the acquired rules. Chapter 11 empirically analyzes the performance of **BAGGER**. The initial inference rules used in the examples are described in appendix A.

Chapter 9

Generalization in BAGGER

The **BAGGER** generalization algorithm is presented in this chapter. This algorithm analyzes explanation structures and detects when a fixed number of applications of some rule can be generalized. It then forms a new sequential rule that supports an arbitrary number of applications of the original rule. For reasons of efficiency, the preconditions of the sequential rule are expressed in terms of the initial state only and do not depend on the results of intermediate applications of the original rule. In addition to describing the algorithm that generalizes the structure of an explanation, this chapter describes how the antecedents of the sequential rule produced are rearranged in order to increase problem-solving performance. A problem solver that applies **BAGGER**'s sequential rules is also described.

9.1. The BAGGER Generalization Algorithm

Figure 9.1 schematically presents how **BAGGER** generalizes the structure of explanations. On the left is the explanation of a solution to some specific problem. In it, some inference rule is repeatedly applied a fixed number of times. In the generalized explanation, the number of applications of the rule is unconstrained. In addition, the properties that must hold in order to

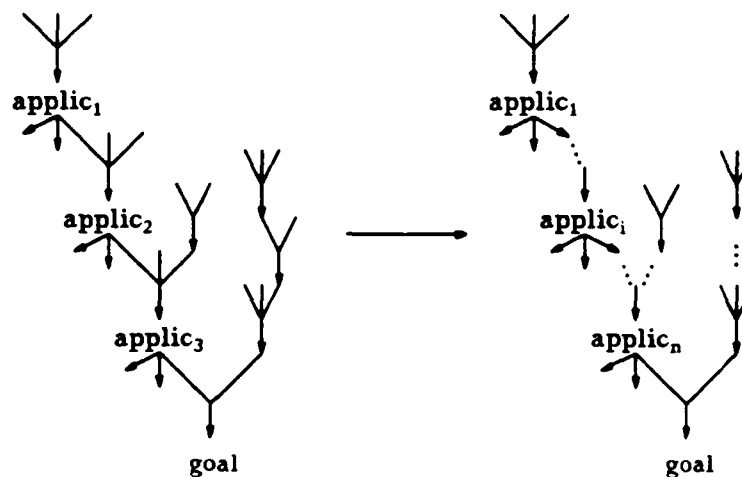


Figure 9.1 Generalizing the Structure of an Explanation

satisfy each application's preconditions, and to meet the antecedents in the goal, are expressed in terms of the initial situation. This means that portions of the explanation not directly involved in the chain of rule applications must also be expressed in terms of the initial state. When the initial situation has the necessary properties, the results of the new rule can be immediately determined, without reasoning about any of the intermediate situations.

The generalization algorithm appears in figure 9.2. This algorithm is expressed in a pseudo-code, while the actual implementation is written in Lisp. The remainder of this section elaborates the pseudo-code. In the algorithm back arrows (\leftarrow) indicate value assignment. The construct

for each element in set do statement

means that *element* is successively bound to each member of *set*, following which *statement* is evaluated. The functions *AddDisjunct* and *AddConjunct* alter their first argument. If either of *AddConjunct*'s arguments is *fail*, its answer is *fail*. *AddRule* places the new rule in the database of acquired rules.

The algorithm begins its analysis of a specific solution at the goal node. It then traces backward, looking for repeated rule applications. To be a candidate, some consequent of one instantiation of a rule must support the satisfaction of an antecedent of another instantiation. These repeated applications need not directly connect — there can be intervening inference rules.

```

procedure BuildNewBAGGERrule(goalNode)
    focusNodes  $\leftarrow$  CollectFocusRuleApplications(goalNode)
    antecedentsInitial  $\leftarrow$  BuildInitialAntecedents(Earliest(focusNodes))
    antecedentsIntermediate  $\leftarrow \phi$ 
    for each focusNode in focusNodes do
        answer  $\leftarrow$  ViewAsArbitraryApplic(focusNode, focusNodes)
        if answer  $\neq$  fail then AddDisjunct(antecedentsIntermediate, answer)
    antecedentsFinal  $\leftarrow$  ViewAsArbitraryApplic(goalNode, focusNodes)
    consequents  $\leftarrow$  CollectGoalTerms(goalNode)
    if antecedentsIntermediate  $\neq \phi \wedge$  antecedentsFinal  $\neq$  fail
        then AddRule(antecedentsInitial, antecedentsIntermediate, antecedentsFinal, consequents)

procedure ViewAsArbitraryApplic (node, focusNodes)
    result  $\leftarrow \phi$ 
    for each antecedent in Antecedents(node) do
        if Axiom?(antecedent) then true
        else if SupportedByEarlierNode?(antecedent, focusNodes) then
            AddConjunct(result, CollectNecessaryEqualities(antecedent, Supporter(antecedent)))
        else if SituationIndependent?(antecedent) then AddConjunct(result, antecedent)
        else if SupportedByPartiallyUnwindableRule?(antecedent) then
            AddConjunct(result, CollectResultsOfPartiallyUnwinding(antecedent))
            AddConjunct(result, ViewAsArbitraryApplic(PartiallyUnwind(antecedent), focusNodes))
        else if SupportedByUnwindableRule?(antecedent) then
            AddConjunct(result, CollectResultsOfUnwinding(antecedent))
        else if SupportedByRuleConsequent?(antecedent) then
            AddConjunct(result, CollectNecessaryEqualities(antecedent, Supporter(antecedent)))
            AddConjunct(result, ViewAsArbitraryApplic(SupportingRule(antecedent), focusNodes))
        else return fail
    return result

```

Figure 9.2 The BAGGER Generalization Algorithm

Once a candidate is found, all the inter-connected instantiations of the underlying general rule are collected.

The general rule repeatedly applied is called a *focus rule*. After a focus rule is found, BAGGER ascertains how an *arbitrary* number of instantiations of this rule and any intervening rules can be concatenated together. This indefinite-length collection of rules is conceptually merged into the explanation, replacing the specific-length collection, and a new rule is produced from the augmented explanation.

A specific solution contains several instantiations of the general rule chosen as the focus rule. Each of these applications of the rule addresses the need of satisfying the rule's antecedents, possibly in different ways. For example, when clearing an object, the blocks moved can be placed in several qualitatively different types of locations. The moved block can be placed on a table (assuming the domain model specifies that tables always have room), it can be placed on a block moved in a previous step, or it can be placed on a block that was originally clear.

BAGGER analyzes all applications of the general focus rule that appear in the specific example. When several instantiations of the focus rule provide sufficient information for different generalizations, BAGGER collects the preconditions for satisfying the antecedents of each in a disjunction of conjunctions (one conjunct for each acceptable instantiation). Common terms are factored out of the disjunction. If none of the instantiations of the focus rule provide sufficient information for generalizing the structure of the explanation, no new rule is learned by BAGGER.

Three classes of terms must be collected to construct the antecedents of a new rule. First, the antecedents of the initial rule application in the arbitrary length sequence of rule applications must be satisfied. To do this, the antecedents of the focus rule are used. Second, the preconditions imposed by chaining together an arbitrary number of rule applications must be collected. These are derived by analyzing each inter-connected instantiation of the focus rule in the sample proof. Those applications that provide enough information to be viewed as the arbitrary *ith* application produce this second class of preconditions. Third, the preconditions from the rest of the explanation must be collected. This determines the constraints on the final applications of the focus rule.

Analyzing the Instantiations of the Focus Rule

In order to package a sequence of rule applications into a single sequential rule, the preconditions that must be satisfied at each of the N rule applications must be collected and combined. The preconditions for applying the resulting extended rule must be specifiable in terms of the initial state, and *not* in terms of intermediate states. This insures, given that the necessary conditions are satisfied in the initial state, a plan represented in a sequential rule will run to completion without further problem solving, regardless of the number of intervening states necessary. For example, there is no possibility that a plan will lead to moving $N-2$ blocks and then get stuck. If the preconditions for the i th rule application were expressed in terms of the result of the $(i-1)$ th application, each of the N rule applications would have to be considered in turn to see if the preconditions of the next are satisfied. This is not acceptable. In the approach taken, extra work during generalization and a possible loss of generality are traded off for a rule whose preconditions are easier to check.

When a focus rule is concatenated an arbitrary number of times, variables need to be chosen for each rule application. The *RIS*, a sequence of p -dimensional vectors, is used to represent this information. The general form of the *RIS* is:

$$\langle v_{1,1}, \dots, v_{1,p} \rangle, \langle v_{2,1}, \dots, v_{2,p} \rangle, \dots, \langle v_n,1, \dots, v_n,p \rangle \quad (9.1)$$

In the tower-building example of figure 8.1, initially $p = 3$: the current situation, the object to be moved, and the object upon which the moved object will be placed.

Depending on the rule used, the choice of elements for this sequence may be constrained. For example, certain elements may have to possess various properties, specific relations may have to hold among various elements, some elements may be constrained to be equal to or unequal to other elements, and some elements may be functions of other elements. Often choosing the values of the components of one vector, determines the values of components of subsequent vectors. For instance, when building a tower, choosing the block to be moved in step i also determines the location to place the block to be moved in step $i+1$.

To determine the preconditions in terms of the initial state, each of the focus rule instantiations appearing in the specific proof is viewed as an arbitrary (or i th) application of the underlying rule. The antecedents of this rule are analyzed as to what must be true of the initial

state in order that it is guaranteed the i th collection of antecedents are satisfied when needed. This involves analyzing the proof tree, considering how each antecedent is proved. A standard explanation-based generalization algorithm, EGGS [Mooney86], is used to determine which variables in the subtree of interest are constrained in terms of other variables in the subtree.

The portion of the proof tree analyzed for a given rule instantiation is that subtree determined by traversing backwards from the given instantiation, collecting nodes until reaching either a leaf node, a situation-independent node, or a node directly supported by a consequent of another focus rule instantiation. Figure 9.3 illustrates this on a section of a hypothetical proof tree. The circles represent focus rule instantiations. The subtree of interest is represented by thicker lines and the given instantiation of the focus rule is represented by a bold circle.

The constrained generalized variables in the subtree of interest are expressed, whenever possible, as components of the p -dimensional vectors described above. The algorithm next

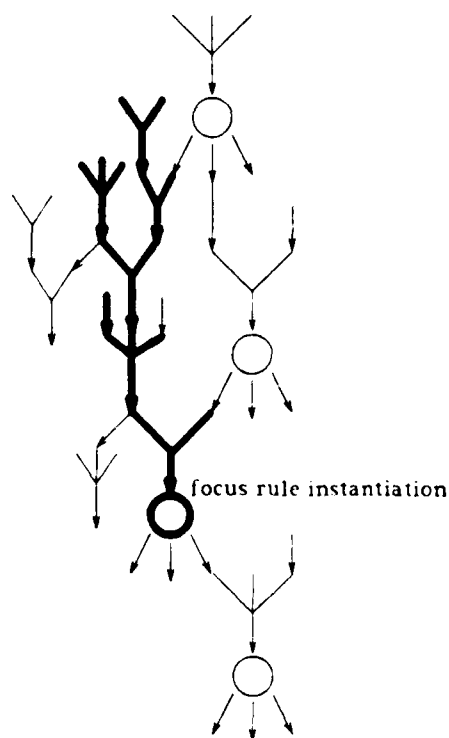


Figure 9.3 Analyzing a Portion of a Proof Tree

ascertains what must be true so that each antecedent is satisfied when necessary. All antecedents of the chosen instantiation of the focus rule must be of one of the following types for generalizing to N to be possible:

- (1) The antecedent may be an *axiom*. Since an axiom always holds, it need not appear as a precondition in the final rule.
- (2) The antecedent may be supported by a consequent of an earlier application of the focus rule. Terms of this type place inter-vector constraints on the sequence of p -dimensional vectors. These constraints are computed by unifying the general versions of the two terms.
- (3) The antecedent may be *situation-independent*. Terms of this type are unaffected by actions.
- (4) The antecedent may be supported by an "unwindable" or partially "unwindable" rule. When this happens, the antecedent is unwound to an arbitrary earlier state and all of the preconditions necessary to insure that the antecedent holds when needed are collected. A *partially unwindable* rule goes back an indefinite number of situations, from which the algorithm continues recursively. If no other inference rules are in the support of the unwindable rule, then it is unwound all the way to the initial state. The processing of unwindable rules is further elaborated later. It, too, may place inter-vector constraints on the sequence of p -dimensional vectors.
- (5) The antecedent is supported by other terms that are satisfied in one of the above ways. When traversing backwards across a supported antecedent, the system collects any inter-vector constraints produced by unifying the general version of the antecedent with the general version of the consequent that supports it.

Notice that antecedents are considered satisfied when they can be expressed in terms of the initial state, and *not* when a leaf of the proof tree is reached. Conceivably, to satisfy these antecedents in the initial state could require a large number of inference rules. If that is the case, it may be better to trace backwards through these rules until more *operational* terms are encountered. This *operationality*/*generality* trade-off [DeJong86, Keller87b, Mitchell86, Segre87b, Shavlik87e] is a major issue in explanation-based learning, but will not be discussed

further in this section. Section 10.2 contains a sequential rule resulting from a more restricted definition of operationality and compares it to the standard sequential rule learned by **BAGGER**. Usually the cost of increased operationality is more limited applicability. An empirical analysis of the effect of this trade-off in the **BAGGER** system appears in chapter 11.

A second point to notice is that not all proof subtrees will terminate in one of the above ways. If this is the case, this application of the focus rule cannot be viewed as an arbitrary *ith* application.¹

The possibility that a specific solution does not provide enough information to generalize to N is an important point in explanation-based approaches to generalizing number. A concept involving an arbitrary number of substructures may involve an arbitrary number of substantially different problems. Any specific solution will only have addressed a finite number of these sub-problems. Due to fortuitous circumstances in the example some of the potential problems may not have arisen. To generalize to N , a system must recognize all the problems that exist in the *general* concept and, by analyzing the specific solution, surmount them. Inference rules of a certain form (described later) elegantly support this task in the **BAGGER** system. They allow the system to reason backwards through an arbitrary number of actions.

The antecedents resulting from the analyzing an application of the focus rule may be satisfied an indefinite number of times when a sequential rule is used. Hence, any non-*RIS* variables in the antecedents produced by the generalization algorithm are added to the *RIS*. These extra vector components can be viewed as "local" variables used when, in the sequential rule, each instantiation of the original focus is being constructed.

Figure 9.4 illustrates how consequents of an earlier application of a focus rule can satisfy some antecedents of a later instantiation. This figure contains a portion of the proof for the tower-building example. (The full proof tree is presented and discussed in section 10.1.) Portions of two consecutive transfers are shown. All variables are universally quantified.

¹ An alternative approach to this would be to have the system search through its collection of unwindable rules and incorporate a relevant one into the proof structure. To study the limits of this thesis' approach to generalizing to N , it is required that *all* necessary information be present in the explanation: no problem-solving search is performed during generalization. Another approach would be to assume the problem solver could overcome this problem at rule application time. This second technique, however, would eliminate the property that a learned plan will always run to completion whenever its preconditions are satisfied in the initial state.

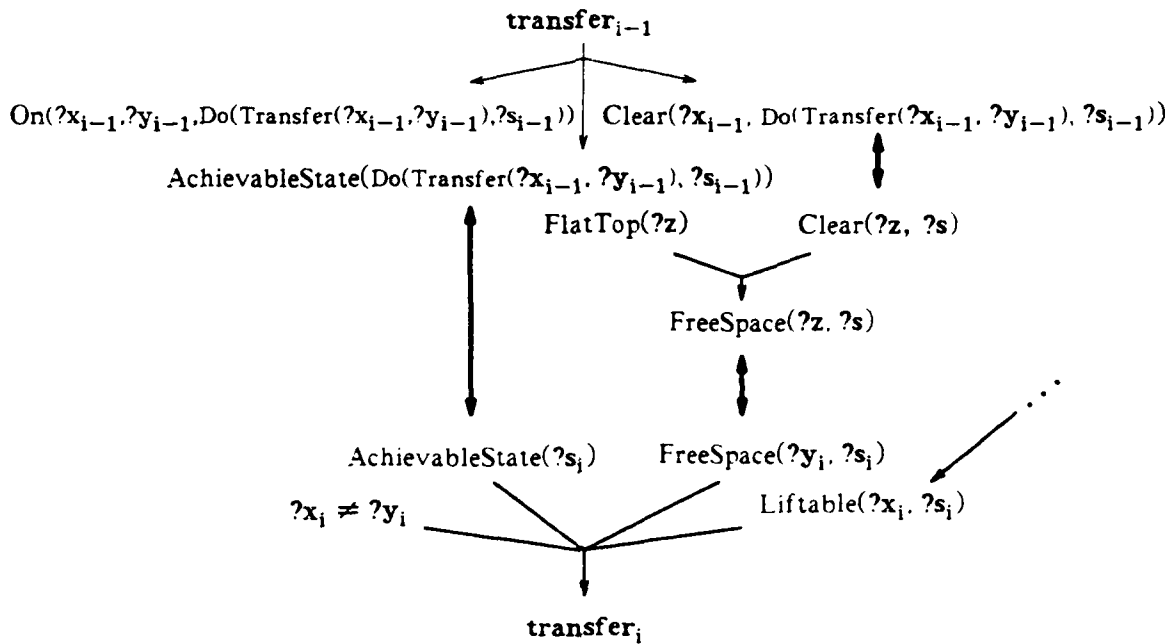


Figure 9.4 Satisfying Antecedents by Previous Consequents

Arrows run from the antecedents of a rule to its consequents. Double-headed arrows represent terms that are equated in the specific explanation. The generalization algorithm enforces the unification of these paired terms, leading to the collection of equality constraints.

There are four antecedents of a transfer. To define a transfer, the block moved (x), the object on which it is placed (y), and the current state (s) must be specified, and the constraints among these variables must be satisfied. One antecedent, the one requiring a block not be placed on top of itself, is type 3 — it is *situation-independent*. The next two antecedents are type 2. Two of the consequents of the $(i-1)th$ transfer are used to satisfy these antecedents of the ith transfer. During $transfer_{i-1}$, in state s_{i-1} object x_{i-1} is moved on to object y_{i-1} . The consequents of this transfer are that a new state is produced, the object moved is clear in the new state, and x_{i-1} is on y_{i-1} in the resulting state.

The state that results from $transfer_{i-1}$ satisfies the second antecedent of $transfer_i$. Unifying these terms defines s_i in terms of the previous variables in the *RIS*.

Another antecedent requires that, in state s_i , there be space on object y_i to put block x_i . This antecedent is type 5, and, hence, the algorithm traverses backwards through the rule that

supports it. An inference rule specifies that a clear object with a flat top has free space. The clearness of x_{i-1} after $transfer_{i-1}$ is used. Unifying this collection of terms leads, in addition to the redundant definition of s_i , to the equating of y_i with z and x_{i-1} . This means that the previously moved block always provides a clear spot to place the current block, which leads to the construction of a tower.

The fourth antecedent, that x_i be liftable, is also type 5. A rule (not shown) states that an object is liftable if it is a clear block. Block x_i is determined to be clear because it is clear in the initial state and nothing has been placed upon it. Tracing backwards from the liftable term leads to several situation-independent terms and the term $Supports(?x_i, \phi, ?s_i)$. Although this term contains a situation variable, it is satisfied by an "unwindable rule," and is type 4.

Equation 9.2 presents the form required for a rule to be unwindable. The consequent must match one of the antecedents of the rule. Hence, the rule can be applied recursively. This feature is used to "unwind" the term from the i th state to an earlier state, often the initial state. Occasionally there can be several unwindable rules in a support path. For example, a block might support another block during some number of transfers, be cleared, remain clear during another sequence of transfers, and finally be added to a tower. An example of a multiple unwinding appears in the next chapter. The variables in the rule are divided into three groups. First, there are the x variables. These appear unchanged in both the consequent's term P and the antecedent's term P . Second, there are the y variables which differ in the two P 's and the z variables that only appear in the antecedents. Finally, there is the state variable (s). There can be additional requirements of the x , y , and z variables (via predicate Q), however, these requirements cannot depend on a state variable.

$$\begin{array}{l}
 P(x_{i,1}, \dots, x_{i,\mu}, y_{i-1,1}, \dots, y_{i-1,\nu}, s_{i-1}) \\
 \text{and} \\
 Q(x_{i,1}, \dots, x_{i,\mu}, y_{i-1,1}, \dots, y_{i-1,\nu}, y_{i,1}, \dots, y_{i,\nu}, z_{i,1}, \dots, z_{i,\omega}) \\
 \text{and} \\
 s_i = Do(x_{i,1}, \dots, x_{i,\mu}, y_{i-1,1}, \dots, y_{i-1,\nu}, z_{i,1}, \dots, z_{i,\omega}, s_{i-1}) \\
 \rightarrow \\
 P(x_{i,1}, \dots, x_{i,\mu}, y_{i,1}, \dots, y_{i,\nu}, s_i)
 \end{array}
 \tag{9.2}$$

Applying equation 9.2 recursively produces equation 9.3. This rule determines the requirements on the earlier state so that the desired term can be guaranteed in state i . Except for the definition of the next state, none of the antecedents depends on the intermediate states. Notice that a collection of y and z variables must be specified. Any of these variables not already contained in the *RIS* are added to it. Hence, the *RIS* is also used to store the results of intermediate computations. Since the predicate Q does not depend on the situation, it can be evaluated in the initial state.

The requirements on the predicate Q are actually somewhat less restrictive. Rather than requiring this predicate to be situation-independent, all that is necessary is that any term containing a situation argument be supported (possibly indirectly) by an application of a focus rule. The important characteristic is that the satisfaction of the predicate Q can be specified in terms of the initial situation only. Separately unwinding a predicate Q while in the midst of unwinding a predicate P is not possible with the current algorithm, and how this can be accomplished is an open research issue.

Frame axioms often satisfy the form of equation 9.2. Figure 9.5 shows one way to satisfy the need to have a clear object at the i th step. Assume the left-hand side of figure 9.5 is a portion of some proof. This explanation says block x_i is clear in state s_i because it is clear in state s_{i-1} and the block moved in $transfer_{i-1}$ is not placed upon x_i . Unwinding this rule leads to the result that block x_i will be clear in state s_i if it is clear in state s_1 and x_i is never used as the new support block in any of the intervening transfers.

$$\begin{array}{l}
 P(x_{i,1}, \dots, x_{i,\mu}, y_{j,1}, \dots, y_{j,\nu}, s_1) \text{ and } 0 < j < i \\
 \text{and} \\
 \forall k \in j+1, \dots, i \\
 \quad Q(x_{i,1}, \dots, x_{i,\mu}, y_{k-j,1}, \dots, y_{k-j,\nu}, y_{k,1}, \dots, y_{k,\nu}, \dots, z_{k,1}, \dots, z_{k,\omega}) \\
 \quad \text{and} \\
 \quad s_k = Do(x_{i,1}, \dots, x_{i,\mu}, y_{k-j,1}, \dots, y_{k-j,\nu}, \dots, z_{k-j,1}, \dots, z_{k-j,\omega}, s_{k-1}) \\
 \rightarrow \\
 P(x_{i,1}, \dots, x_{i,\mu}, y_{i,1}, \dots, y_{i,\nu}, s_i)
 \end{array} \tag{9.3}$$

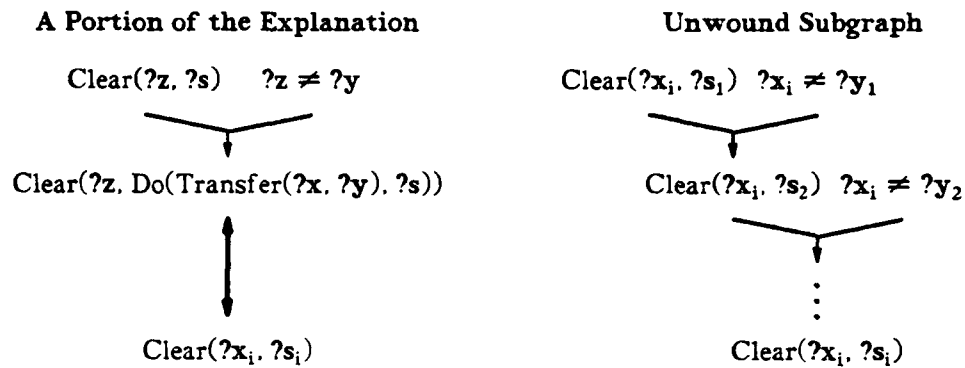


Figure 9.5 Unwinding a Rule

To classify an instantiation of a rule as being unwindable, the rule must be applied at least *twice* successively. This heuristic prevents generalizations that are likely to be spurious. Just like when looking for multiple applications of the focus rule, multiple applications are required for unwindable rules. The intent of this is to increase the likelihood that a generalization is being made that will be prove useful in the future. For example, imagine some rule represents withdrawing some money from a bank and also imagine this rule is of the form of equation 9.2. Assume that in state 5, John withdraws \$500 to buy a television, while in states 1-4, the amount of money he has in the bank is unaffected. While it is correct to generalize this plan to include any number of trips to the bank in order to get sufficient money for a purchase, it does not seem proper to do so. Rather, the generalization should be to a single trip to the bank at *any* time. Frame axioms are exceptions to this constraint — they only need to be applied once to be considered unwindable. Since frame axioms only specify what remains unchanged, there is no risk in assuming an arbitrary number of successive applications.

Incorporating the Rest of the Explanation

Once the repeated rule portion of the extended rule is determined, the rest of the explanation is incorporated into the final result. This is accomplished in the same manner as the way antecedents are satisfied in the repeated rule portion. The only difference is that the focus rule is now viewed as the *Nth* rule application. As before, antecedents must be of one of the five specified types. If all the terms in the goal cannot be satisfied in the arbitrary *Nth* state, no rule is learned.

Determining the Consequents of the New Rule

The consequents of the final rule are constructed by collecting those generalized final consequents of the explanation that directly support the goal.

Chapter 10 discusses the sequential-rules produced by this algorithm when it is applied to several sample problems.

9.2. Problem Solving in BAGGER

A problem solver that applies sequential rules has been implemented. BAGGER's problem solver, in order to construct the *RIS*, is a slightly extended version of a standard backward-chaining problem solver. First, the constraints on $?v_1$ — the initial² vector in the *RIS* — are checked against the initial state, using the standard backward-chaining algorithm. This leads to the binding of other components of the first vector in the sequence. Next, the problem solver checks if the last vector in the sequence (at this point, $?v_i$) satisfies the preconditions for $?v_{n+1}$. If so, a satisfactory sequence has been found and back-chaining terminates successfully. Otherwise, the last vector in the sequence is viewed as $?v_{i-1}$ and the problem solver attempts to satisfy the intermediate antecedent. This may lead to vector $?v_i$ being incorporated into the sequence. If a new vector is added, the final constraints on the sequence are checked again. If they are not satisfied, the new head of the sequence is viewed as $?v_{i-1}$ and the process repeats. This cycle continues until either the current sequence satisfies the rule's antecedents or the initial state cannot support the insertion of another vector into the sequence. When the current sequence cannot be further extended, chronological back-tracking is performed, moving back to the last point where there are choices as how to lengthen the sequence.

Several additional points about BAGGER's problem solver must be made. First, as is standard, arithmetic functions and predicates (e.g., *addition*, *less-than*) are procedurally defined. As discussed in the next section, terms of this type are rearranged so that their arguments are bound when they are evaluated. Second, the antecedents in a rule may involve vectors in addition to $?v_1$, $?v_{i-1}$, $?v_i$, and $?v_{n+1}$. Third, a sequential rule may contain universal and existential terms, where the quantified variable ranges over the vectors in the *RIS*.

² Sometimes, as explained in the next section, the problem solver starts with $?v_{n+1}$ and works backwards to $?v_1$.

In the implementation, the procedure described in the previous paragraph is extended so that, when satisfying the initial antecedents, all the vectors of the form $?v_k$, for fixed k , are instantiated. Similarly, all the vectors of the form $?v_{n-k}$, again for fixed k , are checked to determine termination. In addition, the intermediate antecedents may involve vectors of the form $?v_{i-k}$, for fixed k .

When the quantified variable is a vector in the *RIS*, universal and existential terms can be handled in a brute force manner, due to the finiteness of the *RIS*. If an existential term appears as an antecedent, the problem solver successively binds the quantified variable to vectors in the current *RIS* and attempts to satisfy the body of the term. For universally quantified terms, the quantified variable is set to each vector in the current *RIS*, and the body must be satisfied for each binding. The implemented problem solver does not handle existential terms when the *RIS* is constructed from vector n backwards. (One inefficient way to implement this would be to first ignore the existential terms and construct an otherwise successful *RIS*, then check the existential terms in the manner described above, producing a new *RIS* if checking fails.) In the reverse-construction case, universal terms are satisfied by binding the quantified variable to the earliest vector (the new vector being incorporated) and then viewing every vector as the i th vector.

9.3. Simplifying the Antecedents in Sequential Rules

Even though all the antecedents of a sequential **BAGGER** rule are evaluated in the initial state, substantial time can be spent finding satisfactory bindings for the variables in the rule. Simplifying the antecedents of a rule acquired using EBL can increase the efficiency of the rule [Minton87, Prieditis87]. After a rule is constructed by the **BAGGER** generalization algorithm, duplicate antecedents are removed and the remainder are heuristically rearranged by the system in an attempt to speed-up the process of satisfying the rule.

Simplification in **BAGGER** involves several processes. Heuristics are used to estimate whether is better to construct the *RIS* from the first vector forward or from the last vector backward. Terms not effected by the intermediate antecedent are moved so that they are tested as soon as possible. Intermediate antecedents that are redundantly evaluated are simplified. Terms involving arithmetic are placed so that all their arguments are bound when they are

evaluated. Finally, within each grouping, antecedents are arranged so that terms involving the same variable are grouped near one another.

Determining the Order to Expand the RIS

Since all the antecedents of a sequential rule are specified in terms of the initial state, and not in terms of the results of intermediate states, the *RIS* can be instantiated either from vector 1 to vector n or from vector n to vector 1. However, the order in which new vectors are added to the *RIS* can greatly affect the efficiency of a sequential rule. **BAGGER** contains two heuristics for determining which appears to be the best way to instantiate the *RIS*, given the characteristics of its problem solver.

To see the effect order can have on efficiency, assume that the *RIS* records the height of a tower being planned and that the goal specifies the minimum acceptable tower height. Figure 9.6 illustrates how the tower would be planned, depending on the order the *RIS* is produced. If the *RIS* is constructed by first instantiating the last vector and then proceeding to instantiate the preceding vectors, the final tower height may be chosen first. Next, blocks to be placed in the tower would be selected, in reverse order, each time recording how much of the bottom of the tower remains. At the last step, there may not a suitably-sized block.³ If no collection of movable blocks achieves the chosen final tower height, a new final tower height must be selected and the process repeated. Conversely, if the *RIS* is instantiated from vector 1 forward, much efficiency can be gained. In this example, blocks are selected for insertion in the tower and at

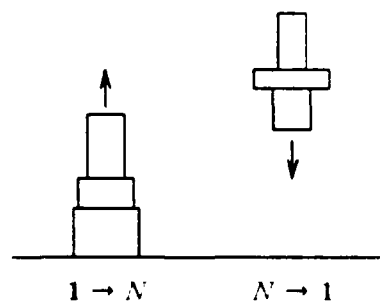


Figure 9.6 Two Ways of Extending an RIS

³ Termination is the topic of the next section. Assume here that once the remaining tower height is negative, the problem solver backtracks. However, **BAGGER**'s problem solver does not operate this cleverly.

each step the current tower height is recorded. Once this height exceeds the goal height, a successful *RIS* is constructed.

For an example where the *RIS* should be constructed from vector n backwards, consider unstacking a tower of blocks in order to clear the block at the bottom of the tower. In this case, which is detailed in the next chapter, the intermediate antecedents in the sequential rule require that the $(i-1)th$ block to be moved support, in the initial state, the $i th$ block to be moved. A final antecedent requires that the last block moved be directly on top of the block to be cleared, while an initial antecedent requires the first block to be moved be clear. If the *RIS* is expanded from vector 1 forward, any clear block will be chosen as the first one to be moved. Next, the blocks underneath it will be successively chosen, each cycle checking to see the $i th$ block to be moved is directly on the block being cleared. If so, a successful *RIS* has been constructed. Otherwise, the expansion continues until no more vectors can be added to the *RIS* and then another initial vector is constructed.

If the sequential rule can be used to clear the desired block, sooner or later a successful *RIS* will be constructed. However, this process can be much more directed if the *RIS* is produced in the reverse direction. The $n th$ vector will contain the block directly on the block to be cleared. The preceding vector will then contain the block directly on the block in the $n th$ vector. This chain will continue until a clear block is incorporated into the *RIS*, satisfying the initial antecedents.

A more sophisticated problem solver than **BAGGER's** could avoid some of these problems. Possibly it could expand an *RIS* both ways from the middle or, if working backwards, leave some of the terms in the final antecedents unsatisfied until the initial antecedents are satisfied. Domain-specific knowledge could also provide additional guidance. However it is desired in this research on learning to minimize the complexity of the problem solver.

A sequential rule is produced from vector n to vector 1 *unless* one of the following occurs:

- (1) The intermediate antecedents contain a *structure-reducing* term whose *decreasing* argument involves a component of vector i and whose *increasing* argument involves the same component of an earlier vector. An example of a structure-reducing term is *RemoveFromBag* ($?x ?y ?z$). The variable $?z$ is the collection that results from removing object $?x$ from the collection $?y$. The decreasing argument is $?z$, while $?y$ is the increasing

argument. The requirements to be a structure-reducing term are that the values of its decreasing argument always be ordered with respect to those of its increasing argument and that there be a smallest possible value of the decreasing argument.

- (2) A term in the final antecedents contains a *threshold* function involving a vector of the form $?v_{n-k}$, for some fixed k . A threshold function is one where if the values of all but one of its arguments are set, there are still an infinite number of acceptable values for the final argument. Sample threshold functions are *less -than* and *greater -than*.

Successfully applying these heuristics means that properties of the some of its predicates must be supplied to BAGGER.

Additional and more sophisticated techniques for determining the most efficient way to construct an *RIS* are needed. The section on open research issues in the conclusion further addresses this topic.

Guaranteeing Termination of *RIS* Expansion

There is the possibility that a problem solver using one of BAGGER's sequential rules can go wandering off along an infinite search tree, as can happen, for example, with depth-first search. Although BAGGER's rules have termination constraints for expanding an *RIS* in either direction (the initial and final antecedents), termination⁴ can be a problem.

For example, if a tower is being built, the *RIS* may be representing the height of the current tower planned. If there is a narrow range of acceptable tower heights, during the construction of the *RIS* the problem solver may "jump" over the acceptable region. Since blocks can only be in a tower once, assuming an finite collection of blocks means the problem solver will terminate, however there is a serious problem looming here. Assume, for instance, that new blocks can always be created. One partial solution may involve reasoning about functions and predicates that deal with ordered arguments (e.g., numbers). For example, if a termination precondition involves the predicate *less -than* and blocks (which must have positive height) are being added, the problem solver can stop before placing all the blocks in a tower. Once the

⁴ That is, termination of the expansion of the *RIS*. The problem solver must also terminate when satisfying each term in the antecedents if it is to successfully instantiate an *RIS*.

tower height recorded in the *RIS* exceeds the goal, there is no reason to continue extending the *RIS*.

Section 10.4 presents an example where reasoning about the predicates in a sequential rule leads to the insertion of another term that reduces the likelihood of time being wasted needlessly expanding a *RIS*. The technique presented is only preliminary, and the problem of inserting new terms to guarantee termination or increase efficiency is an open research issue. Fortunately, the issue of termination has been addressed substantially in program verification research (see [Manna74]). Approaches developed there should prove applicable in restricted situations (the general problem of proving termination — the *halting problem* — is undecidable). The idea of a structure-reducing term discussed in the last section comes from the concept of *well-founded sets*, used in proofs of program termination [Floyd67, Manna70].

Relocating Intermediate Antecedents

Occasionally the intermediate terms may specify redundant constraints on the *RIS*. In addition to removing exact duplicates, the following simplifications are made:

- (1) The intermediate terms may contain $P(?v_{i-1})$ and $P(?v_i)$. This is changed to $P(?v_1)$ and $P(?v_i)$, if the *RIS* is expanded forward, and $P(?v_{i-1})$ and $P(?v_n)$, otherwise.
- (2) The intermediate terms may contain $?v_{i-1,c} = ?v_{i,c}$. In this case, this term is eliminated and all occurrences of $?v_{i-1,c}$ and $?v_{i,c}$ are replaced by $?v_{1,c}$ or $?v_{n,c}$, depending on the direction the *RIS* is constructed. Since BAGGER's generalization algorithm produces many equality constraints, this simplification often proves useful.

Once the direction BAGGER's problem solver will extend a rule's *RIS* is determined, some terms can be moved from one antecedent group to another. If a rule's *RIS* is to be constructed from vector 1 forward, all the terms in the intermediate and final antecedent groups that do not involve vector i nor vector n , are moved to the initial antecedent group. The analogous relocations occur when an *RIS* is constructed from vector n backward. This insures that terms not effected by other portions of the *RIS* are satisfied as early as possible.

Grouping Antecedents

The last attempt to increase efficiency involves rearranging the antecedents within an antecedent group. The algorithm for doing this attempts to locate terms involving the same variables near one another. This algorithm proceeds by first collecting all the variables in the given antecedent group. The variables are next taken from this collection one at a time and placed in a second collection. After each step, all of the terms containing only those variables in the second collection at that point are placed in the new antecedent group. Since some predicates are procedurally defined (e.g., +), a final processing step is needed. All procedurally defined terms are moved to the end of their antecedent group, in order to increase the likelihood that their variables be bound by the time they are evaluated.

9.4. Summary

Most research in explanation-based learning involves relaxing constraints on the variables in an explanation, rather than generalizing the number of inference rules used. This chapter presents a second approach to the task of generalizing the structure of explanations. The approach relies on a shift in representation which accommodates indefinite numbers of rule applications. Compared to the results of standard explanation-based algorithms, rules that are more general are acquired, and since less rules need to be learned, better problem-solving performance gains are achieved.

The fully-implemented **BAGGER** system analyzes explanation structures (in this case, predicate calculus proofs) and detects repeated, inter-dependent applications of rules. Once a rule on which to focus attention is found, the system determines how an *arbitrary* number of instantiations of this rule can be concatenated together. This indefinite-length collection of rules is conceptually merged into the explanation, replacing the specific-length collection of rules, and an extension of a standard explanation-based algorithm produces a new rule from the augmented explanation.

A data structure called an *RIS* is used to record the bindings of variables in the learned rule. The *RIS* accommodates an indefinite number of applications of the focus rule. It also stores intermediate results needed when instantiating the new sequential rule.

Rules produced by BAGGER have the important property that their preconditions are expressed in terms of the initial state — they do not depend on the situations produced by intermediate applications of the focus rule. This means that the results of multiple applications of the rule are determined by reasoning only about the current situation. There is no need to apply the focus rule successively, each time checking if the preconditions for the next application are satisfied.

The specific example guides the extension of the focus rule into a structure representing an arbitrary number of repeated applications. Information not contained in the focus rule, but appearing in the example, is often incorporated into the extended rule. In particular, *unwindable* rules provide the guidance as to how preconditions of the *i*th application can be specified in terms of the current state.

A concept involving an arbitrary number of substructures may involve any number of substantially different problems. However, a specific solution will have necessarily only addressed a finite number of them. To generalize to N , a system must recognize all the problems that exist in the general concept and, by analyzing the specific solution, surmount them. If the specific solution does not provide enough information to circumvent all general problems, generalization to N cannot occur because BAGGER is designed not to perform any problem-solving search during generalization. When a specific solution surmounts, in an extendible fashion, a sub-problem in different ways during different instantiations of the focus rule, disjunctions appear in the acquired rule.

Once a new sequential rule is produced, its antecedents are rearranged in an attempt to increase its problem-solving performance. This involves several processes. Whether to construct an *RIS* from the first vector forward or from the last vector backward is determined heuristically. Terms not effected by the intermediate antecedents are moved so that they are tested as soon as possible, while intermediate antecedents that are redundantly evaluated are simplified. Terms that are procedurally defined are placed so that all their arguments are bound when they are evaluated. Finally, antecedents are arranged so that terms involving the same variable are grouped near one another.

The next chapter presents the details of several examples encountered by BAGGER. Complete proof trees and the learned sequential rules are presented. In addition, some

interesting properties and current shortcomings of the generalization algorithm are illustrated. An empirical analysis of the benefit of generalizing the structure of explanations is reported in chapter 11.

Chapter 10

Details of Several BAGGER Examples

This section presents the details of **BAGGER**'s acquisition of several sequential rules. Besides providing the details of the examples mentioned previously, plus a few others, several interesting aspects of the **BAGGER** generalization algorithm are illustrated. For instance, unlike other work in explanation-based learning, a rule learned by **BAGGER** does not always subsume the problem solution from which it is derived. Other topics discussed include how extra terms can be added to a sequential rule in order to increase its efficiency, how the process of unwinding may limit the generality of an acquired sequential rule, how **BAGGER** performs on problems not involving situation calculus, and how the complexity of the sample solution can effect the generality of the rule learned.

The first section in this chapter presents in great detail how a rule for building a blocks-world tower is learned. The reason for each term appearing in the final result is described. Much less discussion is provided for the remaining examples. Only their major points are covered.

Following the description of the tower-building rule, several other rules acquired from tower-building examples are discussed. The result obtained by a standard explanation-based generalization algorithm is presented, for purposes of comparison. Also presented is the result obtained from a more operational version of **BAGGER**, in which no nodes are pruned from explanations. Finally, a more general rule that results from a slightly more complicated sample problem is described.

Next, there are several examples involving the acquisition of plans for clearing objects. Two different examples are covered, plus an example where **BAGGER** does not acquire the fully general concept underlying the example's solution. The reason for this is discussed. Following that there are two examples involving non-blocks world domains. A plan for setting a table and a rule that represents a general version of DeMorgan's law. The first of these illustrates the need for adding additional terms to a sequential rule, while the second (which is not expressed in situation calculus) involves the reformulation of a two-input version of DeMorgan's law into an N -input version.

As stated in the previous chapter, after a rule is initially produced by **BAGGER**, antecedents are rearranged by the system in order to speed-up the process of satisfying the rule. While this reordering means the presented rules are somewhat harder to read, they accurately reflect the rules acquired and used by the system.

10.1. Building a Tower

The proof that explains the tower-building actions of figure 8.1, where three blocks are stacked on top of one another, appears in figure 10.1. This graph, and those that follow, are produced by the **BAGGER** system, however nodes have been rearranged by hand for the sake of readability. Since the situation arguments are quite lengthy, they are abbreviated and a key appears in the figure. Arrows run from the antecedent of a rule to its consequent. When a rule has multiple antecedents or consequents, an ampersand (&) is used. Descriptions of all the rules used in this structure are contained in appendix A. The primed ampersands are the instantiations of the focus rule, while the ampersand nearest the bottom of the graph is the goal node.

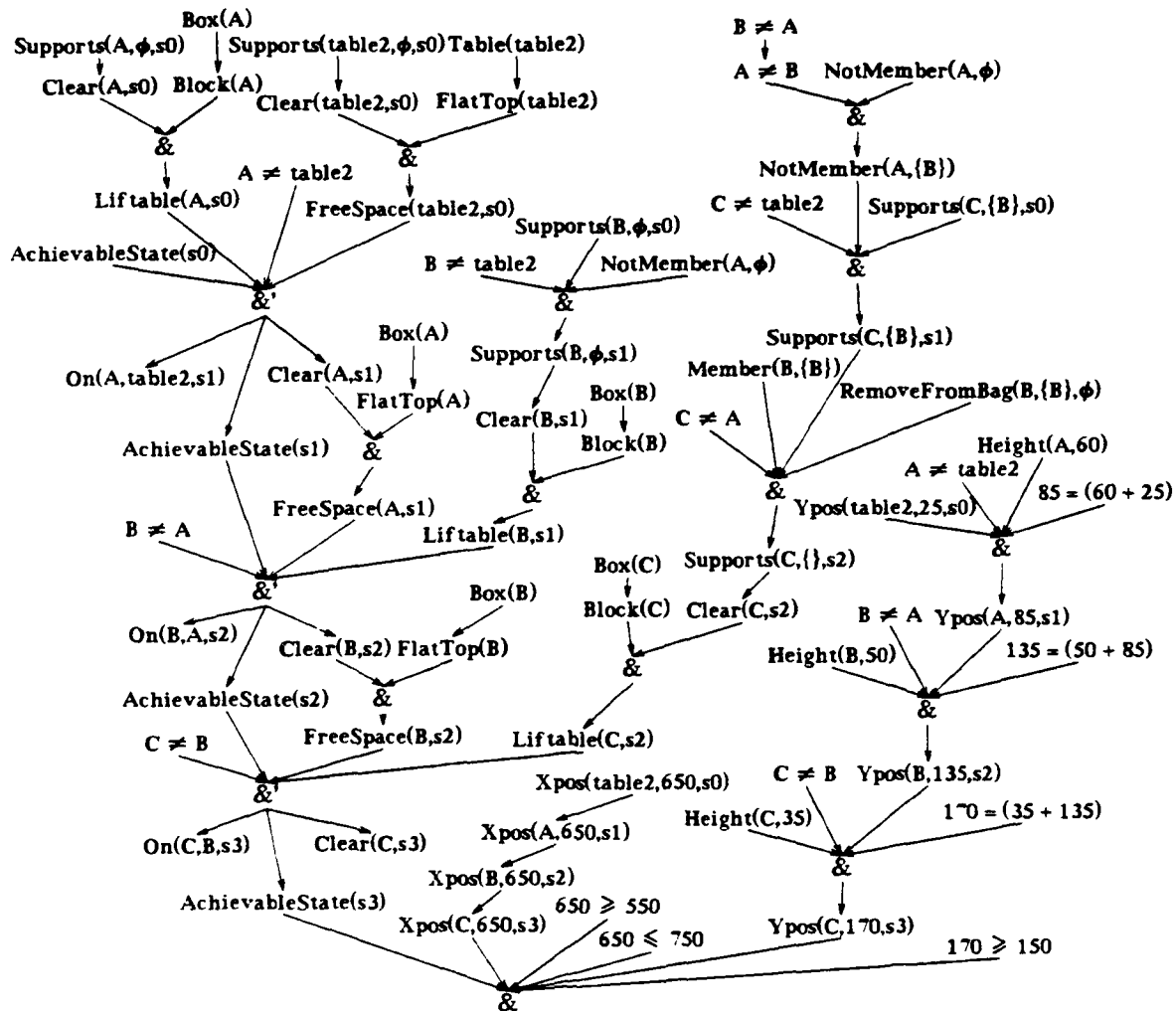


Figure 10.1 Situation Calculus Plan for Stacking Three Blocks

Abbreviation Key

s0	the initial state	s2	Do(Transfer(B,A),Do(Transfer(A,table2),s0))
s1	Do(Transfer(A,table2),s0)	s3	Do(Transfer(C,B),Do(Transfer(B,A),Do(Transfer(A,table2),s0)))

The goal provided to the backward-chaining theorem prover that produced this graph is:

$$\begin{aligned} \exists \quad & \text{AchievableState}(\text{?state}) \quad \wedge \\ & \text{Xpos}(\text{?object}, \text{?px}, \text{?state}) \quad \wedge \quad \text{?px} \geq 550 \quad \wedge \quad \text{?px} \leq 750 \quad \wedge \\ & \text{Ypos}(\text{?object}, \text{?py}, \text{?state}) \quad \wedge \quad \text{?py} \geq 150. \end{aligned}$$

This says that the goal is to prove the existence of an achievable state, such that in that state the horizontal position of some object is between 550 and 750 and the vertical position of that same objects is at least 150.

The sequential-rule **BAGGER** produces by analyzing this explanation structure appears in table 10.1. The remainder of this section describes how each term in this table is produced. Line numbers have been included for purposes of reference. For readability, the new rule is broken down into components, as shown in equation 10.1.

Producing the Initial Antecedents

The initial antecedents in the first line of the rule establish a sequence of vectors, the initial state, and the first vector contained in the sequence. Subscripts are used to indicate components of vectors, as a shorthand for functions that perform this task. For example, $?v_{1,3}$ is shorthand for *ThirdComponent*($?v_1$). Lines 2 and 3 contain the antecedents of the first application in the chain of applications. These are the same terms that appear in the focus rule (the first rule in table A.3), except that the components of v_1 are used. The system has knowledge of which arguments are situation variables and the initial state constant $s0$ is placed in these positions. The other terms in this grouping are produced by the unwinding process (*Height*, *Xpos*, *Ypos*, and the addition term) or are moved (\geq and \leq) from the final antecedents to the initial antecedents because their variables are not influenced by the intermediate antecedents. The terms produced by unwinding are described further in what follows.

Analyzing the Applications of the Focus Rule

Lines 5-11 contain the preconditions derived by analyzing the three instantiations of the focus rule. In this implication, v_i - an arbitrary vector in the sequence (other than the first) - is used, as these constraints must be satisfied for each of the applications that follow the first. Vector v_{i-1} is the vector immediately preceding v_i . It is needed because some of the antecedents of the i th application are satisfied by the $(i-1)$ th application. Although some preconditions in

Table 10.1 The Components of the Learned Rule

Antecedents_{initial}

- (1) Sequence(?seq) \wedge InitialVector(?v₁,?seq) \wedge State(s0) \wedge ?v_{1,1} = s0 \wedge
- (2) FreeSpace(?v_{1,3}, s0) \wedge Liftable(?v_{1,2}, s0) \wedge Height(?v_{1,2}, ?v_{1,4}) \wedge
- (3) Xpos(?v_{1,3}, ?px, s0) \wedge Ypos(?v_{1,3}, ?new, s0) \wedge ?v_{1,2} \neq ?v_{1,3} \wedge
- (4) ?v_{1,5} = (?v_{1,4} + ?new) \wedge ?px \geq ?xmin \wedge ?px \leq ?xmax

Antecedent_{intermediate}

- (5) [Member(?v_i, ?seq) \wedge ?v_i \neq ?v₁ \wedge Member(?v_{i-1}, ?seq) \wedge Predecessor(?v_{i-1}, ?v_i, ?seq)
→
- (6) ?v_{i,3} = ?v_{i-1,2} \wedge ?v_{i,1} = Do(Transfer(?v_{i-1,2}, ?v_{i-1,3}), ?v_{i-1,1}) \wedge FlatTop(?v_{i,3}) \wedge
- (7) Block(?v_{i,2}) \wedge Height(?v_{i,2}, ?v_{i,4}) \wedge ?v_{i,2} \neq ?v_{i,3} \wedge ?v_{i,5} = (?v_{i,4} + ?v_{i-1,5})
- (8) [[[Member(?v_j, ?seq) \wedge Earlier(?v_j, ?v_i, ?seq) \rightarrow ?v_{i,2} \neq ?v_{j,3}] \wedge Supports(?v_{i,2}, ϕ , s0)]
- (9) \vee [[Member(?v_j, ?seq) \wedge Earlier(?v_j, ?v_{i-1}, ?seq) \rightarrow NotMember(?v_{j,2}, {?v_{i-1,2}})] \wedge
- (10) [Member(?v_j, ?seq) \wedge Earlier(?v_j, ?v_{i-1}, ?seq) \rightarrow ?v_{i,2} \neq ?v_{j,3}] \wedge
- (11) Supports(?v_{i,2}, {?v_{i-1,2}}, s0) \wedge ?v_{i,2} \neq ?v_{i-1,3}]]]

Antecedents_{final}

- (12) FinalVector(?v_n, ?seq) \wedge ?py = ?v_{n,5} \wedge ?state = Do(Transfer(?v_{n,2}, ?v_{n,3}), ?v_{n,1}) \wedge
- (13) ?object = ?v_{n,2} \wedge ?py \geq ?ymin

Consequents

- (14) State(?state) \wedge Xpos(?object, ?px, ?state) \wedge ?px \leq ?xmax \wedge ?px \geq ?xmin \wedge
- (15) Ypos(?object, ?py, ?state) \wedge ?py \geq ?ymin

This rule extends sequences 1 \rightarrow N.

$$\text{Antecedents}_{\text{initial}} \wedge \text{Antecedent}_{\text{intermediate}} \wedge \text{Antecedents}_{\text{rest}} \rightarrow \text{Consequents} \quad (10.1)$$

the new rule involve v_i and v_{i-1} , these preconditions all refer to conditions in the initial state. They do *not* refer to results in intermediate states.

The final two of the three instantiations of the focus rule produce sufficient information to determine how the antecedents of the rule can be satisfied in the *i*th application. In the first application (upper left of figure 10.1), neither the support for *Liftable* nor the support for *FreeSpace* provide enough information to determine the constraints on the initial state so that these terms can be satisfied in an arbitrary step. In both cases, the proof only had to address clearness in the current state. No information is provided within the proof as to how clearness can be guaranteed to hold in some later state.

Two different ways of satisfying the antecedents are discovered by analyzing the two other instantiations of the focus rule, and, hence, a disjunction is learned. The common terms in these two disjuncts appear in lines 6 and 7, while the remaining terms for the first disjunction are in line 8 and for the second in lines 9-11.

The third term in line 7 is the vector form of the inequality that is one of the antecedents of the focus rule. This, being situation-independent, is a type 3 antecedent. In vector form, it becomes $v_{i,2} \neq v_{i,3}$. It constrains possible collections of vectors to those that have different second and third members. This constraint stems from the requirement that a block cannot be stacked on itself.

Both of the successful applications of the focus rule have their *AchievableState* term satisfied by a consequent of a previous application. These terms are type 2 and require collection of the equalities produced by unifying the general versions of the matching consequents and antecedents. (See figure 9.4 for the details of these matchings.) The equality that results from these unifications is the second term of line 6. Thus, the next state is always completely determined by the previous one. No searching needs to be done in order to choose the next state. (Actually, no terms are ever evaluated in these intermediate states. The only reason they are recorded is so that the final state can be determined, for use in setting the situation variable in the consequents.)

Both successful applications have their *FreeSpace* term satisfied in the same manner. Traversing backwards across one rule leads to a situation independent term (*FlatTop* - line 6) and the consequent of an earlier application (*Clear*). Unifying the two clear terms (again, see

figure 9.4) produces the first two equalities in line 6. This first equality means that the block to be moved in the i th step can always be placed on top of the block to be moved in the $(i-1)$ th step. No problem solving need be done to determine the location at which to continue building the tower.

The *Block* term in line 7 is produced during the process of analyzing the way the *Liftable* term is satisfied. The remaining portion of the analysis of *Liftable* produces the terms in the disjunctions. As in the initial antecedents, the *Height* and addition terms in line 7 are produced during the analysis of the terms in the goal, which is described later.

In the second application of the focus rule, which produces the first disjunct, a clear block to move is acquired by finding a block that is clear because it supports nothing in the initial state and nothing is placed on it later. The frame axiom supporting this is an unwindable rule. Unwinding it to the initial state produces line 8. The *Supports* term must hold in the initial state and the block to be moved in step i can never be used as the place to locate a block to be moved in an earlier step. The general version of the term *NotMember* (A, ϕ) does not appear in the learned rule because it is an axiom that nothing is a member of the empty set and axioms are pruned. (An earlier unification, from the rule involving *Clear*, requires that the second variable in the general version of *NotMember* term be ϕ .)

Notice that this unwinding, which leads to a more operational rule because it produces preconditions entirely expressed in terms of the initial state, restricts the applicability of the acquired rule. The first disjunct requires that if an initially clear block is to be added to the tower, nothing can ever be placed on it, even temporarily. A more general plan would be learned, however, if in the specific example a block is temporarily covered. In that case, in the proof there would be several groupings of unwindable rules: for awhile the block would remain clear, something would then be placed on it and it would remain covered for several steps, and finally it would be cleared and remain that way until moved. Although this clearing and unclearing can occur repeatedly, the BAGGER algorithm is unable to generalize number within unwindable subproofs.

The second disjunct (lines 9-11) results from the different way a liftable block is found in the third application of the focus rule. Here a liftable block is found by using a block that initially supported one other block, which is moved in the previous step, and where nothing else

is moved to the lower block during an earlier rule application. Unwinding the subgraph for this application leads to the requirements that initially one block is on the block to be moved in step i , that block be moved in step $(i-1)$, and nothing else is scheduled to be moved to the lower block during an earlier rule applications. Again, some terms do not appear in the learned rule (*Member* and *RemoveFromBag*) because, given the necessary unifications, they are axioms. This time *NotMember* is not an axiom, and hence, appears. If the specific example were more complicated, the acquired rule would reflect the fact that the block on top can be removed in some *earlier* step, rather than necessarily in the *previous* step. This case is discussed in the section 10.2.

Analyzing the Rest of the Explanation

Once all of the instantiations of the focus rule are analyzed, the goal node is visited. This produces lines 12 and 13, plus some of the earlier terms.

The *AchievableState* term of the goal is satisfied by the final application of the focus rule, leading to the third term in line 12.

The final *Xpos* is calculated using an unwindable rule. Tracing backwards from the *Xpos* in the goal to the consequent of the unwindable rule produces the first term in line 13, as well as the third term in line 12. When this rule is unwound it produces the first term of line 3 and the second term of line 6. Also, matching the *Xpos* term in the antecedents with the one in the consequents, so that equation 2 applies, again produces the first term in line 6. Since there are no "Q"-terms (equation 9.3), no other preconditions are added to the intermediate antecedent.

The inequalities involving the tower's horizontal position are state-independent; their general forms are moved to the initial antecedents because their arguments are not effected by satisfying the intermediate antecedent. These terms in the initial antecedents involving *?px* insure that the tower is started underneath the goal.

Unwindable rules also determine the final *Y*-position. Here "Q"-terms are present. The connection of two instantiations of the underlying general rule appears in figure 10.2. This general rule is unwound to the initial state, which creates the second term of line 3 and the second term of line 6. The last three terms of line 7 are also produced, as the "Q"-terms must

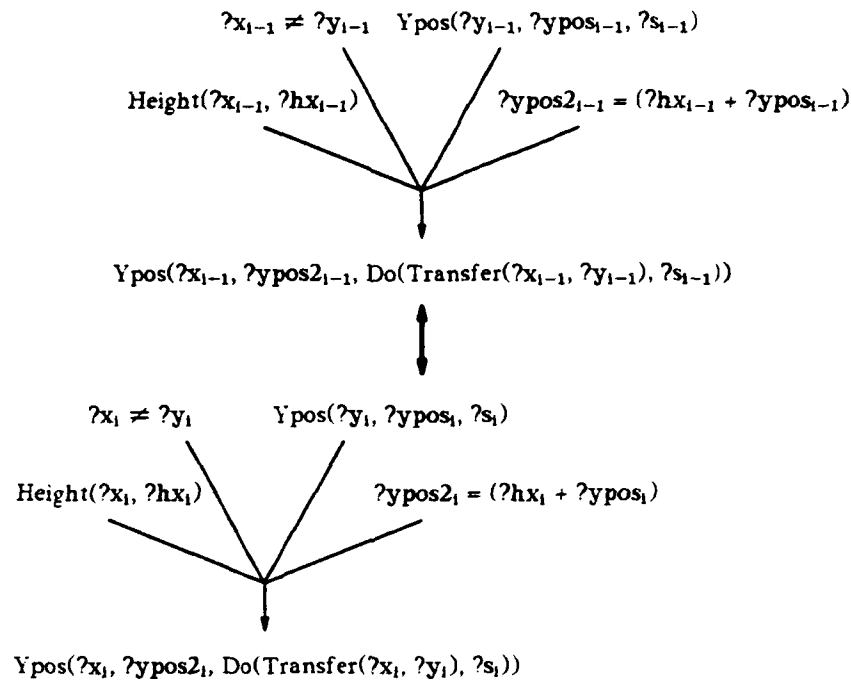


Figure 10.2 Calculating the Vertical Position of the i th Stacked Block

hold for each application of the unwound rule. This process adds two components to the vectors in the *RIS*. The first ($?v_{i,4}$) comes from the $?hx$ variable, which records the height of the block being added to the tower. The other ($?v_{i,5}$) comes from the variable $?yPos2$. It records the vertical position of the block added, and hence, represents the height of the tower. The $?ypos$ variable does not lead to the creation of another *RIS* entry because it matches the $?yPos2$ variable of the previous application. All that is needed is a $?ypos$ variable for the first application. Similarly, matching the $Ypos$ term in the antecedents with the one in the consequents produces the first term in line 6.

The last conjunct in the goal produces the second term on line 13. This precondition insures that the final tower is tall enough.

Finally, the general version of the goal description is used to construct the consequents of the new rule (lines 14 and 15).

10.2. More Tower Building Rules

The examples in this section consider some variants on the tower-building rule. First, the results of applying standard explanation-based generalization to the same proof tree are presented. Next, again using the same proof, a more operational rule is produced. Finally, using a slightly more complicated example, the acquisition of a more general rule is described.

The Results of Standard Explanation-Based Learning

The result of directly applying the standard explanation-based generalization algorithm EGGS [Mooney86] to the proof in figure 10.1 appears in table 10.2. This rule specifies how to move from the initial state to the state that results from three transfers, where the moved blocks are stacked upon one another, creating the desired tower. Notice that this rule requires the first and second blocks to be moved be initially clear, while the third block to be moved must only support the second block to be moved. This rule cannot be used to construct a tower

Table 10.2 The Standard EBL Version of the Tower Rule

Antecedents
(1) $\text{AchievableState}(s_0) \wedge \text{Liftable}(\text{?x2}, s_0) \wedge \text{FlatTop}(\text{?x2}) \wedge \text{Height}(\text{?x2}, \text{?hx}) \wedge$
(2) $\text{Block}(\text{?x}) \wedge \text{Height}(\text{?x}, \text{?hx2}) \wedge \text{?x2} \neq \text{?x} \wedge \text{FreeSpace}(\text{?y}, s_0) \wedge \text{Xpos}(\text{?y}, \text{?xpos}, s_0) \wedge$
(3) $\text{Ypos}(\text{?y}, \text{?pyy}, s_0) \wedge \text{?x} \neq \text{?y} \wedge \text{?x2} \neq \text{?y} \wedge \text{Supports}(\text{?x}, \{\text{?x1}\}, s_0) \wedge$
(4) $\text{Not.Member}(\text{?x2}, \{\text{?x1}\}) \wedge \text{Supports}(\text{?x1}, \phi, s_0) \wedge \text{Block}(\text{?x1}) \wedge \text{FlatTop}(\text{?x1}) \wedge$
(5) $\text{Height}(\text{?x1}, \text{?hx1}) \wedge \text{?x1} \neq \text{?y} \wedge \text{?x2} \neq \text{?x1} \wedge \text{?x} \neq \text{?x1} \wedge \text{?xpos} \leq \text{?xmax} \wedge$
(6) $\text{?xpos} \geq \text{?xmin} \wedge \text{?pyx} = (\text{?hx} + \text{?pyy}) \wedge \text{?pyx1} = (\text{?hx1} + \text{?pyx}) \wedge$
(7) $\text{?pyx2} = (\text{?hx2} + \text{?pyx1}) \wedge \text{?pyx2} \geq \text{?ymin}$
Consequents
(8) $\text{AchievableState}(\text{Do}(\text{Transfer}(\text{?x}, \text{?x1}), \text{Do}(\text{Transfer}(\text{?x1}, \text{?x2}), \text{Do}(\text{Transfer}(\text{?x2}, \text{?y}), s_0)))) \wedge$
(9) $\text{Xpos}(\text{?x}, \text{?xpos}, \text{Do}(\text{Transfer}(\text{?x}, \text{?x1}), \text{Do}(\text{Transfer}(\text{?x1}, \text{?x2}), \text{Do}(\text{Transfer}(\text{?x2}, \text{?y}), s_0)))) \wedge$
(10) $\text{?xpos} \leq \text{?xmax} \wedge \text{?xpos} \geq \text{?xmin} \wedge$
(11) $\text{Ypos}(\text{?x}, \text{?pyx2}, \text{Do}(\text{Transfer}(\text{?x}, \text{?x1}), \text{Do}(\text{Transfer}(\text{?x1}, \text{?x2}), \text{Do}(\text{Transfer}(\text{?x2}, \text{?y}), s_0)))) \wedge$
(12) $\text{?pyx2} \geq \text{?ymin}$

from a scene with three blocks all directly on a table, nor can it be used to move a three-block tower from one location to another.

Chapter 11 presents an empirical determination of the performance gains obtained by using the sequential rules learned by **BAGGER** rather than the rules learned by standard explanation-based generalization.

A More Operational Tower Building Rule

An issue in explanation-based learning is deciding which portions of a specific problem's explanation can be considered easily reconstructable and, hence, disregarded when the explanation is generalized. Allowing such reconstruction during problem solving produces a more *general* rule since alternative reconstructions are possible, but the resulting rule is less *operational* because significant effort can be expended recalculating. As described there, in **BAGGER** a term is considered easily evaluated if it is expressed in terms of the initial state only. A more restricted definition, one that incorporates all of the constraints in the explanation, is to consider a node acceptable only if it is a leaf node in the explanation. Being a leaf node means it is either an axiom or is a term used to specify the initial state. In latter case, it is reasonable to assume that these terms will also be specified in future problems and will require minimal effort to test.

BAGGER can be instructed to construct sequential rules that are more operational. This entails forming the initial antecedents by collecting the leaf nodes that support the first application of the focus rule¹ and altering the algorithm for constructing the remainder of the sequential rule to only accept a situation-independent rule if it is a leaf node. The results of using this more operational form of generalization with the previous tower building proof (figure 10.1) appears in table 10.3. Terms appearing in this rule, but not in the more general version (table 10.1), are in all capitals and in a bold font.

Comparing the two versions, a clear table² is required for the location to place the first block moved, as opposed to a possibly hard to find arbitrary free space. Also, rather than

¹ This could be easily extended to view *each* application of the focus rule as the first and using the disjunction of all the results to construct the initial antecedents.

² Although a known rule says that tables are always clear, that rule is not used in the explanation of the specific example.

Table 10.3 The More Operational Version of the Tower Rule

Antecedents _{initial}	
(1)	Sequence(?seq) \wedge InitialVector(?v ₁ , ?seq) \wedge State(s0) \wedge ?v _{1,1} = s0 \wedge
(2)	TABLE(?v _{1,3}) \wedge SUPPORTS(?v _{1,3} , ϕ , s0) \wedge BOX(?v _{1,2}) \wedge SUPPORTS(?v _{1,2} , ϕ , s0) \wedge
(3)	Height(?v _{1,2} , ?v _{1,4}) \wedge Xpos(?v _{1,3} , ?px, s0) \wedge Ypos(?v _{1,3} , ?new, s0) \wedge ?v _{1,2} \neq ?v _{1,3} \wedge
(4)	?v _{1,5} = (?v _{1,4} - ?new) \wedge ?px \geq ?xmin \wedge ?px \leq ?xmax
Antecedent _{intermediate}	
(5)	[Member(?v _j , ?seq) \wedge ?v _j \neq ?v ₁ \wedge Member(?v _{i-1} , ?seq) \wedge Predecessor(?v _{i-1} , ?v _j , ?seq) →
(6)	?v _{i,3} = ?v _{i-1,2} \wedge ?v _{i,1} = Do(Transfer(?v _{i-1,2} , ?v _{i-1,3} , ?v _{i-1,1}) \wedge BOX(?v _{i,3}) \wedge
(7)	BOX(?v _{i,2}) \wedge Height(?v _{i,2} , ?v _{i,4}) \wedge ?v _{i,2} \neq ?v _{i,3} \wedge ?v _{i,5} = (?v _{i,4} + ?v _{i-1,5})
(8)	[[[Member(?v _j , ?seq) \wedge Earlier(?v _j , ?v _j , ?seq) → ?v _{i,2} \neq ?v _{j,3}] \wedge Supports(?v _{i,2} , ϕ , s0)]
(9)	\vee [[Member(?v _j , ?seq) \wedge Earlier(?v _j , ?v _{i-1} , ?seq) → Not.Member(?v _{j,2} , {?v _{i-1,2} })] \wedge
(10)	[Member(?v _j , ?seq) \wedge Earlier(?v _j , ?v _{i-1} , ?seq) → ?v _{i,2} \neq ?v _{j,3}] \wedge
(11)	Supports(?v _{i,2} , (?v _{i-1,2}), s0) \wedge ?v _{i,2} \neq ?v _{i-1,3}]]]
Antecedents _{final}	
(12)	FinalVector(?v _n , ?seq) \wedge ?py = ?v _{n,5} \wedge ?state = Do(Transfer(?v _{n,2} , ?v _{n,3} , ?v _{n,1}) \wedge
(13)	?object = ?v _{n,2} \wedge ?py \geq ?ymin
Consequents	
(14)	State(?state) \wedge Xpos(?object, ?px, ?state) \wedge ?px \leq ?xmax \wedge ?px \geq ?xmin \wedge
(15)	Ypos(?object, ?py, ?state) \wedge ?py \geq ?ymin
<i>This rule extends sequences 1 → N.</i>	

specifying any liftable object, a clear box is specified as the first object to be moved. The new requirements on successive moved blocks are that each must be a box, which is a specialization of block. Finally, the previously moved object must be a box, as that insures it will have a flat top.

A More General Tower Building Rule

The rules learned by BAGGER depend on the complexity of the specific problem from which they are derived. If the specific problem's solution took advantage of some fortuitous circumstances, some issues inherent in the underlying general concept may not have been addressed. Because BAGGER performs no additional problem solving during generalization, the rule acquired may not fully reflect the possible generality. One way this can occur is if each application of the focus rule is satisfied in the same manner, even though alternatives are possible. In this case, a disjunctive rule will not be produced. This section also illustrates this issue with a slightly more complicated tower-building example. A rule more general than that produced from the previous example results.

Figure 10.3 contains the initial and final states of the example, where four blocks are stacked to reach a goal height. The relevant portion of the resulting proof tree appears in figure 10.4. This subtree shows how it is determined that block *D* can be lifted. Some rules are numbered, using subscripts on their ampersands, for purposes of reference. The sequential rule that results is nearly identical to that in table 10.1, with one major difference. The second disjunct in that rule (lines 9-11) is replaced with the existential term in table 10.4. This term says that it is possible to plan to move a block in the *ith* step if it originally supports only one other block, which is to be moved in some *earlier* step, and no other block is to be placed on it in

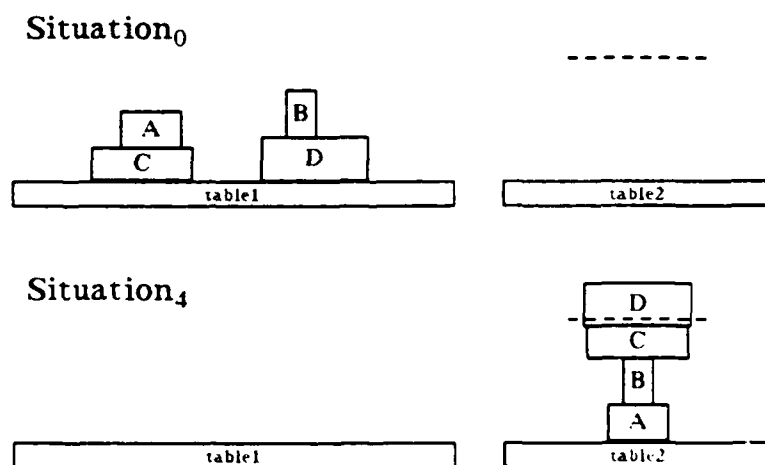


Figure 10.3 Constructing a Four-Block Tower

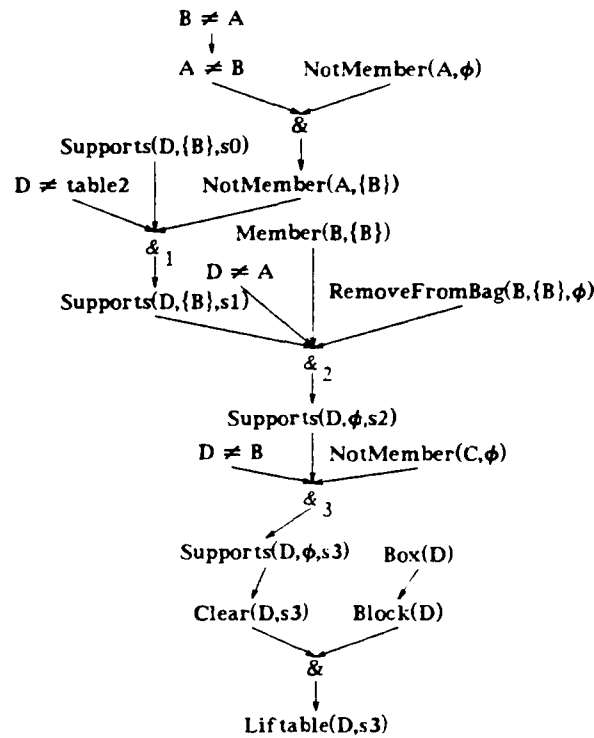


Figure 10.4 Partial Situation Calculus Plan for Stacking Four Blocks

Abbreviation Key

s0	the initial state	s2	Do(Transfer(B,A),Do(Transfer(A,table2),s0))
s1	Do(Transfer(A,table2),s0)	s3	Do(Transfer(C,B),Do(Transfer(B,A),Do(Transfer(A,table2),s0)))

any intervening step. This is more general than the other plan, which said a block could be moved if the only block on top of it is moved in the *previous* step.

The proof tree in figure 10.4 contains two connected unwindable rules, which explains why the existential term is produced. The rule in that figure subscripted with a three is unwindable. When unwound, it says a block continues to support nothing as long as the block moved in each step is not placed upon it (producing line 4 in the table). This gets unwound to rule 2, which states that moving the only block on an object clears it. The first unwound rule is unwound to an arbitrary step k , so this clearing occurs in step $k-1$. Next, rule 1 is unwound. This rule instantiation says an object supporting only one other object stays that way as long as

Table 10.4 A More General Way to Find a Liftable Block

$\exists ?v_k \in ?seq$

- (1) $\text{NotEarlier}(?v_i, ?v_k, ?seq) \wedge ?v_{i,2} \neq ?v_{k-1,3} \wedge \text{Supports}(?v_{i,2}, \{?v_{k-1,2}\}, s0) \wedge$
 - (2) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_{k-1}, ?seq) \rightarrow ?v_{i,2} \neq ?v_{j,3}] \wedge$
 - (3) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_{k-1}, ?seq) \rightarrow \text{NotMember}(?v_{j,2}, \{?v_{k-1,2}\})] \wedge$
 - (4) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_i, ?seq) \wedge \text{NotEarlier}(?v_j, ?v_k, ?seq) \rightarrow ?v_{i,2} \neq ?v_{j,3}]$
-

nothing else is set on it. Unwinding it to the initial state produces the third term in line 1, plus the terms in lines 2 and 3.

This example illustrates the need for recognizing when a newly-acquired rule subsumes an previously learned rule. If this can be done, redundant rules can be removed from the database. A final point is that additional analysis of an explanation, for example, considering the insertion of extra unwindable rules, may prove beneficial in acquiring the general concept underlying a sample solution.

10.3. Clearing an Object

This section presents three examples dealing with the clearing of blocks. The first two produce orthogonal plans for clearing blocks, while the third demonstrates why **BAGGER** does not learn a plan combining the approaches in the first two.

Unstacking a Tower

The acquisition of the rule earlier illustrated by figure 8.4 is presented in this section. The sequence of moves from which a plan for unstacking towers is learned appears in figure 10.5. Here three blocks are moved to clear block *D*. The explanation of these transfers appears in figure 10.6, while the rule that results appears in table 10.5.

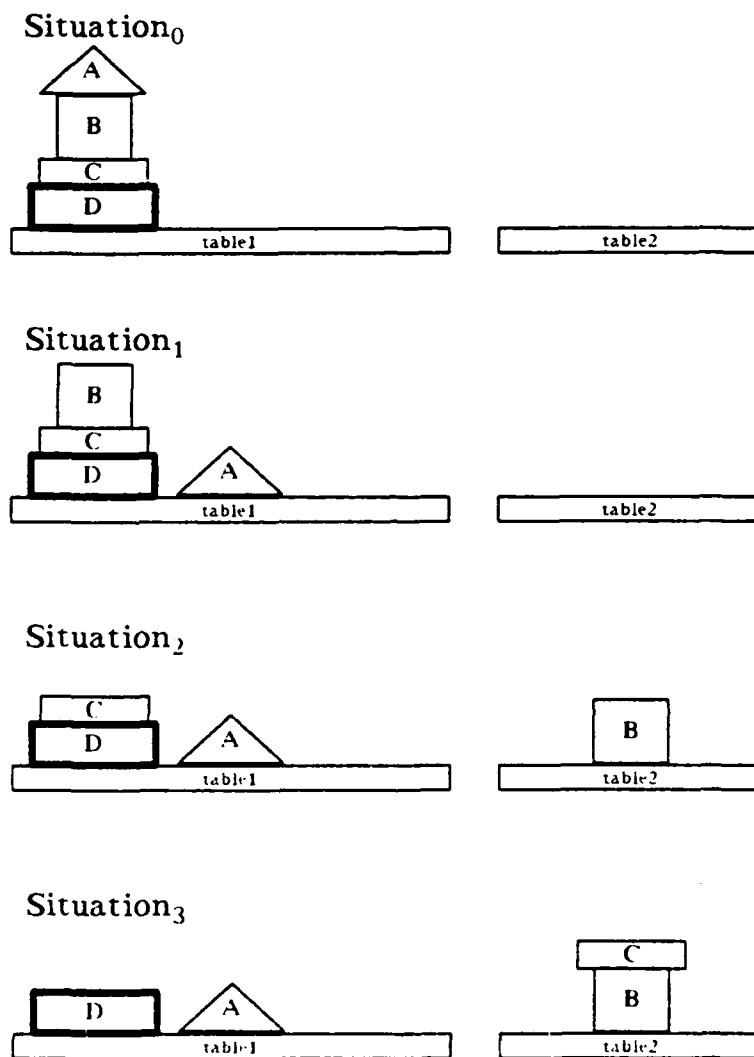


Figure 10.5 Moving Three Blocks to Clear Another Block

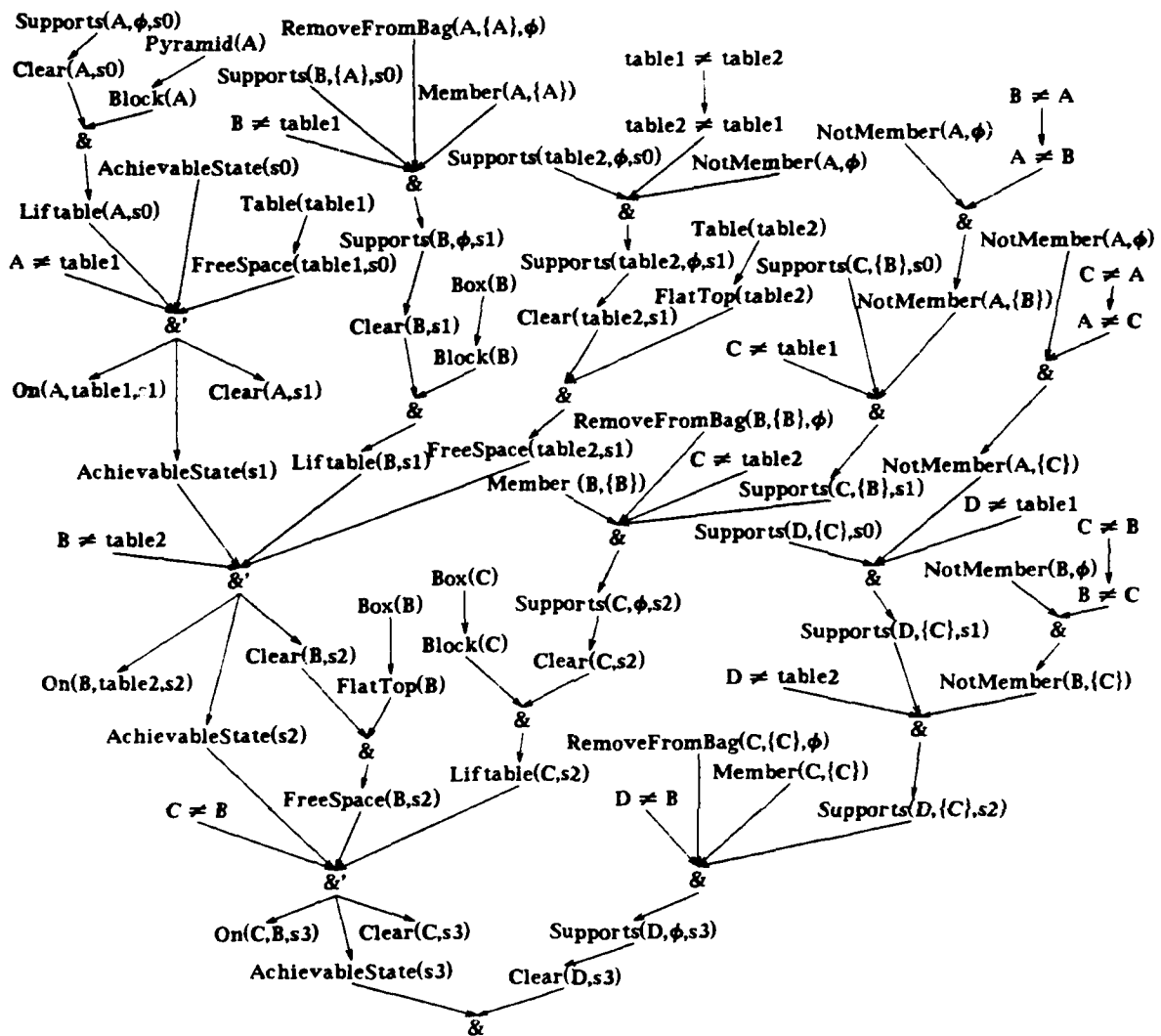


Figure 10.6 Situation Calculus Plan for Clearing Block D

Abbreviation Key

s0 the initial state	s2 Do(Transfer(B, table2), Do(Transfer(A, table1), s0))
s1 Do(Transfer(A, table1), s0)	s3 Do(Transfer(C, B), Do(Transfer(B, table2), Do(Transfer(A, table1), s0)))

Table 10.5 Unstacking a Tower

Antecedents _{initial}	
(1)	$\text{InitialVector}(\text{?v}_1, \text{?seq}) \wedge \text{State}(\text{s0}) \wedge \text{?v}_{1,1} = \text{s0} \wedge \text{FreeSpace}(\text{?v}_{1,3}, \text{s0}) \wedge$
(2)	$\text{Liftable}(\text{?v}_{1,2}, \text{s0}) \wedge \text{?v}_{1,2} \neq \text{?v}_{1,3}$
Antecedent _{intermediate}	
(3)	$[\text{Member}(\text{?v}_i, \text{?seq}) \wedge \text{?v}_i \neq \text{?v}_1 \wedge \text{Member}(\text{?v}_{i-1}, \text{?seq}) \wedge \text{Predecessor}(\text{?v}_{i-1}, \text{?v}_i, \text{?seq})$ \rightarrow
(4)	$\text{FlatTop}(\text{?v}_{i,3}) \wedge \text{Block}(\text{?v}_{i,2}) \wedge \text{?v}_{i,2} \neq \text{?v}_{i,3} \wedge \text{?v}_{i,3} = \text{?v}_{i-1,2} \wedge$
(5)	$\text{?v}_{i,1} = \text{Do}(\text{Transfer}(\text{?v}_{i-1,2}, \text{?v}_{i-1,3}), \text{?v}_{i-1,1}) \wedge \text{Supports}(\text{?v}_{i,2}, \{\text{?v}_{i-1,2}\}, \text{s0}) \wedge$
(6)	$\text{?v}_{i,2} \neq \text{?v}_{i-1,3} \wedge [\text{Member}(\text{?v}_j, \text{?seq}) \wedge \text{Earlier}(\text{?v}_j, \text{?v}_{i-1}, \text{?seq}) \rightarrow \text{?v}_{i,2} \neq \text{?v}_{j,3}] \wedge$
(7)	$[\text{Member}(\text{?v}_j, \text{?seq}) \wedge \text{Earlier}(\text{?v}_j, \text{?v}_{i-1}, \text{?seq}) \rightarrow \text{NotMember}(\text{?v}_{j,2}, \{\text{?v}_{i-1,2}\})] \wedge$
(8)	$\text{?object} \neq \text{?v}_{i-1,3} \wedge \text{NotMember}(\text{?v}_{i-1,2}, \{\text{?v}_{n,2}\})]$
Antecedents _{final}	
(9)	$\text{Sequence}(\text{?seq}) \wedge \text{FinalVector}(\text{?v}_n, \text{?seq}) \wedge \text{?state} = \text{Do}(\text{Transfer}(\text{?v}_{n,2}, \text{?v}_{n,3}), \text{?v}_{n,1}) \wedge$
(10)	$\text{Supports}(\text{?object}, \{\text{?v}_{n,2}\}, \text{s0}) \wedge \text{?object} \neq \text{?v}_{n,3}$
Consequents	
(11)	$\text{State}(\text{?state}) \wedge \text{Clear}(\text{?object}, \text{?state})$
<i>This rule extends sequences $N \rightarrow I$.</i>	

This rule's *RIS*, which only contains the variables in the focus rule, is satisfied from N to 1. The last term in line 6 and the first in line 10 require that the block to be moved in step $i-1$ must be the only one directly upon the block to be moved in step i . In addition, the block to be moved in step i is placed upon the block to be moved in the previous step (last term of line 4). Hence, a tower of blocks is cleared by inverting the tower at another location. Once a location to place the first block is calculated, no more work is expended finding acceptable locations to place the other blocks.

There is one additional interesting point about this example. Unlike the results of most other learning algorithms, the learned rule does not apply to the sample problem which

motivated it! In the acquired rule, an object is cleared by inverting the stack of blocks upon it at another location. However, this will not work if the block on top of the stack does not have a flat top, as is the case in the sample problem.

In the **BAGGER** generalization algorithm, each application of the focus rule is analyzed. Only those that are suitably general provide information that is incorporated into the intermediate antecedents of the acquired rule. If an application performs some operation in an overly-specific way, the acquired rule will not reflect this nuance and the rule's generality may be limited.

If the sample problem is more complicated, **BAGGER** learns a disjunctive rule for clearing. For example, this will occur if a tower is moved and some of the intermediate blocks are placed on a table (which are known to always have free space) and others on some other blocks that are clear in the initial state. In these situations, the learned rule would subsume the example of figure 10.5.

Horizontally Clearing an Object

Unlike many other blocks-world formalizations, the rules used by **BAGGER** do not assume a block can support only one other block. Figure 10.7 contains the results of a sequence of moves from which another plan (figure 8.5) for clearing an object is acquired. Blocks *A* and *B* are moved in order to clear table 1. Here, the clearing is called "horizontal" because in general

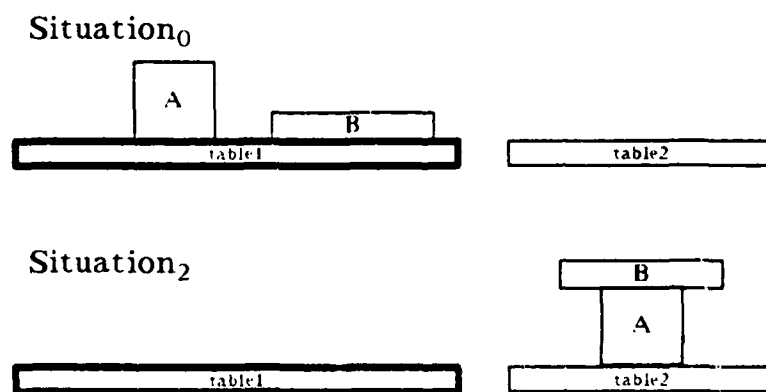


Figure 10.7 Moving Two Blocks to Clear a Table

it involves clearing an object that is supporting any number of clear objects. This compares to the previous example of "vertical" clearing, where a stack of objects is moved.

In figure 10.8, the explanation of these transfers appears. Table 10.6 contains the rule that results. In this new acquired rule, the initial antecedents determine which objects are being supported in the initial state by the object that is to be cleared. One of these is chosen as the first object to be moved. At each successive step, another object in the collection is chosen to be moved and is scheduled to be placed upon the block to be moved in the previous step. This continues until the collection is empty, at which time the goal will be achieved. The final term in line 8 insures that no block is moved on to the object to be cleared. The *RIS*'s vectors contains one variable in addition to the three in the focus rule. This fourth variable records how many blocks will still be on the block being cleared after the *i*th step.

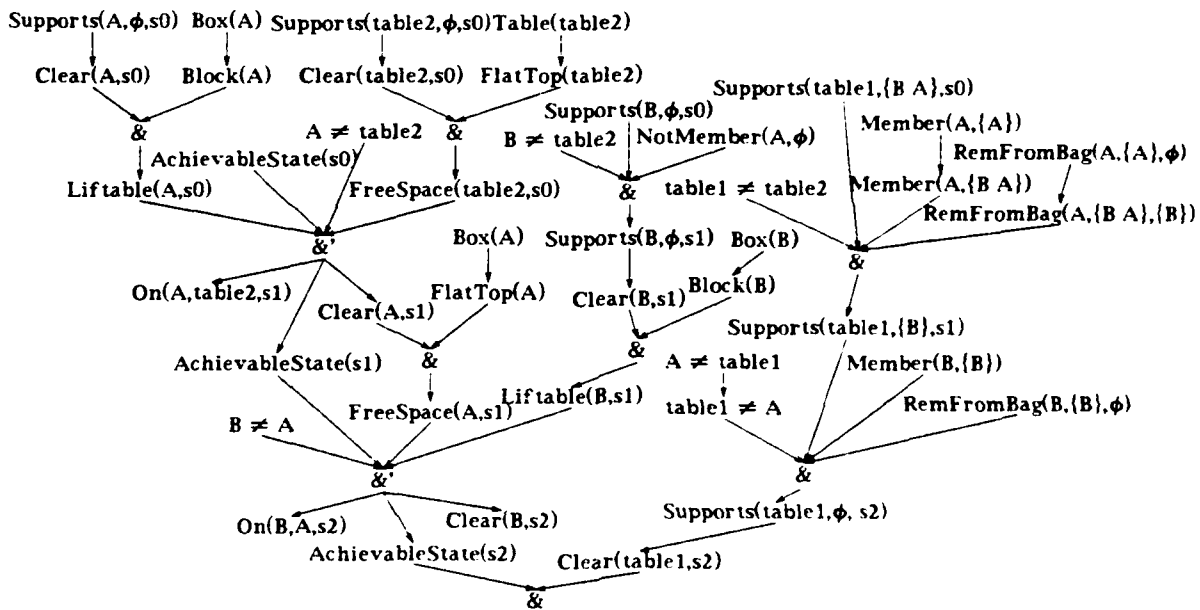


Figure 10.8 Situation Calculus Plan for Clearing Table 1

Abbreviation Key

s0	the initial state
s1	Do(Transfer(A, table2), s0)
s2	Do(Transfer(B, table2), Do(Transfer(A, table2), s0))

Table 10.6 "Horizontally" Clearing an Object

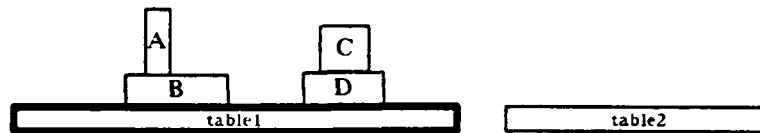
Antecedents _{initial}	
(1)	Sequence(?seq) \wedge InitialVector(?v ₁ , ?seq) \wedge State(s0) \wedge ?v _{1,1} = s0 \wedge
(2)	FreeSpace(?v _{1,3} , s0) \wedge Litable(?v _{1,2} , s0) \wedge ?v _{1,2} \neq ?v _{1,3} \wedge ?object \neq ?v _{1,3} \wedge
(3)	Member(?v _{1,2} , ?new) \wedge RemoveFromBag(?v _{1,2} , ?new, ?v _{1,4}) \wedge Supports(?object, ?new, s0)
Antecedent _{intermediate}	
(4)	[Member(?v _i , ?seq) \wedge ?v _i \neq ?v ₁ \wedge Member(?v _{i-1} , ?seq) \wedge Predecessor(?v _{i-1} , ?v _i , ?seq) →
(5)	?v _{i,3} = ?v _{i-1,2} \wedge ?v _{i,1} = Do(Transfer(?v _{i-1,2} , ?v _{i-1,3} , ?v _{i-1,1}) \wedge
(6)	Member(?v _{i,2} , ?v _{i-1,4}) \wedge RemoveFromBag(?v _{i,2} , ?v _{i-1,4} , ?v _{i,4}) \wedge FlatTop(?v _{i,3}) \wedge
(7)	Block(?v _{i,2}) \wedge Supports(?v _{i,2} , ϕ , s0) \wedge ?v _{i,2} \neq ?v _{i,3} \wedge
(8)	[Member(?v _j , ?seq) \wedge Earlier(?v _j , ?v _i , ?seq) → ?v _{i,2} \neq ?v _{j,3}] \wedge ?object \neq ?v _{i,3}]
Antecedents _{final}	
(9)	FinalVector(?v _n , ?seq) \wedge ?v _{n,4} = ϕ \wedge ?state = Do(Transfer(?v _{n,2} , ?v _{n,3} , ?v _{n,1})
Consequents	
(10)	State(?state) \wedge Clear(?object, ?state)
<i>This rule extends sequences I → N.</i>	

A Less than Fully Successful Example

Much can be understood about an algorithm by seeing examples where the algorithm does not produce the proper result. An example where BAGGER does not acquire the fully general underlying concept is presented in this section. The example combines the vertical and horizontal clearing seen in the last two examples. While the sequential rule acquired is valid, it does not fully represent the generality underlying the problem solution from which it is learned.

An example involving moving four blocks to clear a table appears in figure 10.9. This can be viewed as moving two towers while vertically clearing an object. The ability of the plan that

Situation₀



Situation₄



Figure 10.9 Moving Two Towers to Clear a Table

results is illustrated by figure 10.10. Here, two towers containing an arbitrary number of blocks can be moved in order to clear a supporting object. However, this plan is not as general as it could be. The algorithm should have also generalized the number of towers, rather than requiring there be exactly two. The problem of handling multiple generalizations to N is beyond the current capabilities of the **BAGGER** system, largely because only a single *RIS* is constructed. Incorporating multiple *RIS*'s in one rule is an open research issue.

The portion of the proof tree that demonstrates table 1 is clear after the four block movements appears in figure 10.11. Similar to a previous example (figure 10.4), multiple unwindable rules are in the proof. The rule subscripted with a 1 says an object supporting two

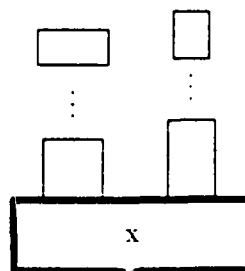


Figure 10.10 Type of Objects that the New General Plan can Clear

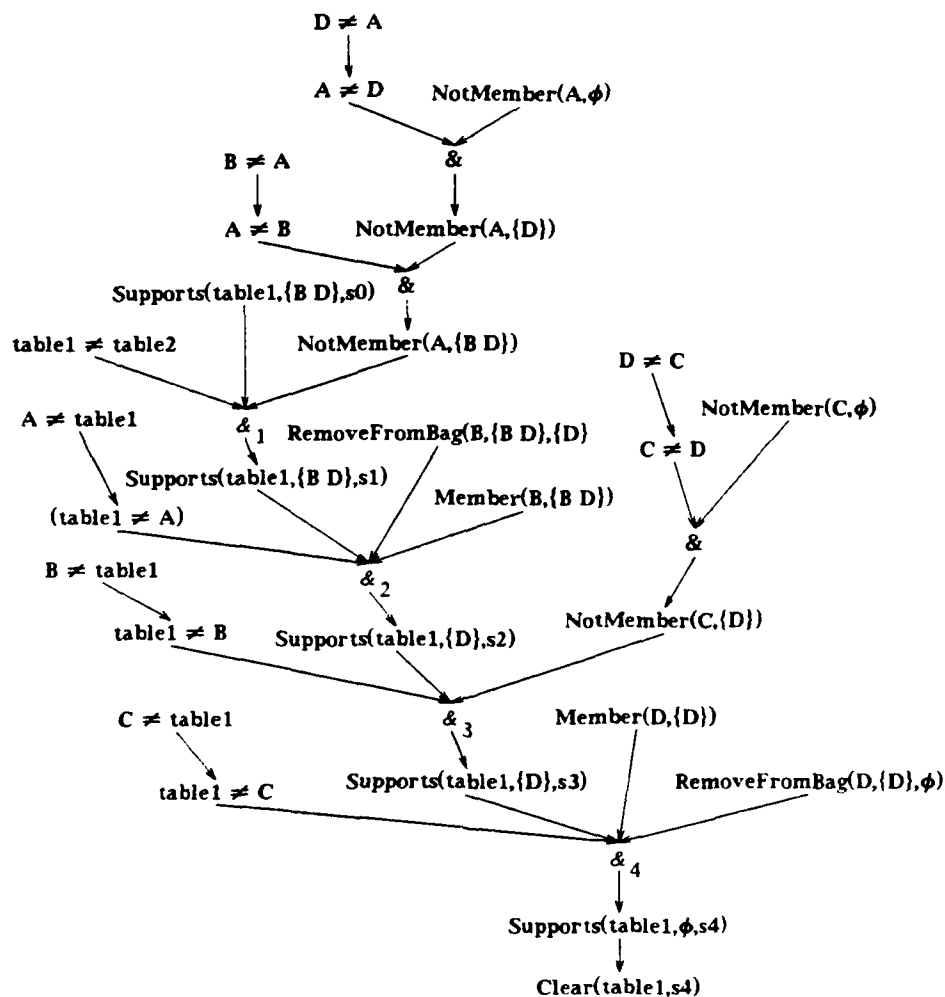


Figure 10.11 Partial Situation Calculus Plan for Moving Two Towers

Abbreviation Key

- s0 *the initial state*
- s1 Do(Transfer(A, table2), s0)
- s2 Do(Transfer(B, A), Do(Transfer(A, table2), s0))
- s3 Do(Transfer(C, B), Do(Transfer(B, A), Do(Transfer(A, table2), s0)))
- s4 Do(Transfer(D, C), Do(Transfer(C, B), Do(Transfer(B, A), Do(Transfer(A, table2), s0))))

blocks stays that way for awhile, rule 2 removes one block, rule 3 says the object continues to support one block, and rule 4 records the clearing of the table.

Table 10.7 contains a portion of the final antecedents of the acquired rule. This existential term requires that the object to be cleared support two objects in the initial state, one which is the last object moved and the other which is moved sometime earlier. The intermediate terms for this sequential rule are similar to those of the tower-building rule (table 10.1). They state that the blocked to be moved in step i must be either clear in the initial state or initially support the block to be moved in step $i-1$. This insures, if the rule is be successfully satisfied, that all the blocks on the two blocks selected by the final antecedents are moved.

The reason that this rule does not fully represent the general concept is that BAGGER does not detect the pattern in the unwinding where *each* of the blocks on the table are moved. Instead, this is seen as moving *two* blocks to clear an object. A possible solution to this shortcoming is to extend BAGGER so that it can be applied recursively during generalization.

Table 10.7 A Portion of a Plan to Clear a Block Supporting Two Towers

- $$\exists ?v_k \in ?seq$$
- (1) $\text{NotEarlier}(?v_n, ?v_k, ?seq) \wedge ?object \neq ?v_{k-1,3} \wedge \text{Supports}(?object, \{?v_{k-1,2} ?v_{n,2}\}, s0) \wedge$
 - (2) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_{k-1}, ?seq) \rightarrow ?object \neq ?v_{j,3}] \wedge$
 - (3) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_{k-1}, ?seq) \rightarrow \text{NotMember}(?v_{j,2}, \{?v_{k-1,2} ?v_{n,2}\})] \wedge$
 - (4) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_n, ?seq) \wedge \text{NotEarlier}(?v_j, ?v_k, ?seq) \rightarrow ?object \neq ?v_{j,3}] \wedge$
 - (5) $[\text{Member}(?v_j, ?seq) \wedge \text{Earlier}(?v_j, ?v_n, ?seq) \wedge \text{NotEarlier}(?v_n, ?v_k, ?seq) \rightarrow \text{NotMember}(?v_{j,2}, \{?v_{n,2}\})]$
-

10.4. Setting a Table

Occasionally much effort will be expended by **BAGGER**'s problem solver trying to construct an *RIS* that satisfies the current goal before it determine all possible variable bindings lead to failure. This section presents an example in which additional analysis can lead to the determination of when it will be futile to try to instantiate a satisfactory *RIS*. From this example, a plan for setting N places on a table is produced. Additional analysis by **BAGGER** determines that it is futile to attempt to apply this rule unless the capacity of the table is at least N .

The graph and rule presented in this section are produced by **BAGGER**. However, the algorithm for constructing the extra terms is unappealingly narrow and is not discussed outside of this section. The technique is intended to only illustrate the kind of post-processing that is necessary to improve the efficiency of the rules learned by **BAGGER**. In the system, this technique is implemented within a production system architecture. There are a collection of rules that can add new terms into a sequential rule. These rules continually add new terms until none of the rules have their preconditions satisfied.

The example from which the new rule is acquired is sketched in figure 10.12. Initially a table with a seating capacity of four is empty. First one, then another, set of utensils is placed on this table in satisfying the goal of finding a table with two locations set.

The proof that the goal is satisfied by the actions in figure 10.12 appears in figure 10.13 (some long predicate names are abbreviated in this graph). A known rule states that to set a place on a table, there must be a clear spot on the table and there must be a set of utensils available. This rule also states that after setting the empty place it is no longer empty and the previously free utensils are now in use. A second rule is used to count the number of table locations that are set in a given situation.

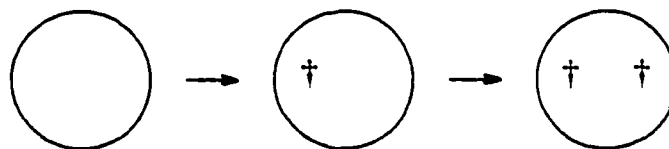


Figure 10.12 Setting Two Places at a Table

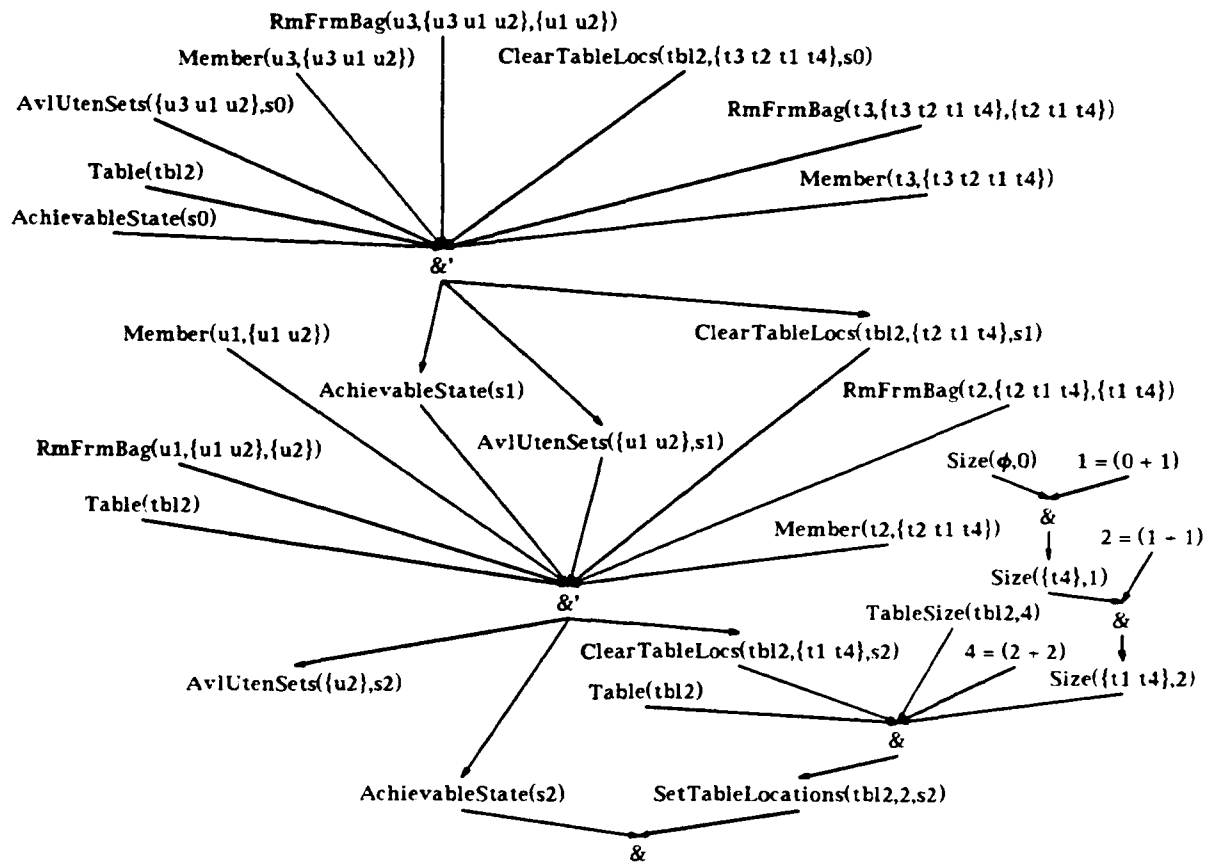


Figure 10.13 Situation Calculus Plan for Setting Two Places at a Table

Abbreviation Key

- s_0 the initial state
 s_1 $\text{Do}(\text{SetTblLoc}(\text{tbl2}, \text{t3}, \text{u3}), s_0)$
 s_2 $\text{Do}(\text{SetTblLoc}(\text{tbl2}, \text{t2}, \text{u1}), \text{Do}(\text{SetTblLoc}(\text{tbl2}, \text{t3}, \text{u3}), s_0))$

The rule that BAGGER produces by generalizing the explanation in figure 10.13 is contained in table 10.8. The meaning of each of the eight components in this rule's *RIS* appears in table 10.9. Basically, the rule records, in the initial situation, the open locations on some table and the number of available utensils sets. Then utensil sets are placed on open table locations until enough locations are set.

However, if there are not enough locations on the table or not enough utensil sets, substantial useless work may be performed. By analyzing the initially-produced sequential rule, **BAGGER** adds the last term in line 6, which requires that the size of the table chosen be sufficient to set the desired number of places.

Table 10.8 A Rule for Setting Tables

Antecedents_{initial}

- (1) Sequence(?seq) \wedge InitialVector(?v₁, ?seq) \wedge AchievableState(s0) \wedge
 - (2) AvailUtensilSets(?v_{1,3}, s0) \wedge ?v_{1,6} = {?v_{1,7} | ?v_{1,8}} \wedge ?v_{1,3} = {?v_{1,4} | ?v_{1,5}} \wedge
 - (3) ?table = ?v_{1,1} \wedge ?v_{1,2} = s0 \wedge RemoveFromBag(?v_{1,7}, ?v_{1,6}, ?v_{1,8}) \wedge
 - (4) Member(?v_{1,7}, ?v_{1,6}) \wedge ClearTableLocs(?v_{1,1}, ?v_{1,6}, s0) \wedge
 - (5) RemoveFromBag(?v_{1,4}, ?v_{1,3}, ?v_{1,5}) \wedge Member(?v_{1,4}, ?v_{1,3}) \wedge Table(?v_{1,1}) \wedge
 - (6) TableSize(?v_{1,1}, ?size) \wedge ?settings \leq ?size
-

Antecedent_{intermediate}

- (7) [Member(?v_i, ?seq) \wedge ?v_i \neq ?v₁ \wedge Member(?v_{i-1}, ?seq) \wedge Predecessor(?v_{i-1}, ?v_i, ?seq)
 \rightarrow
 - (8) ?v_{i,6} = ?v_{i-1,8} \wedge ?v_{i,3} = ?v_{i-1,5} \wedge
 - (9) ?v_{i,2} = Do(SetTblLoc(?v_{1,1}, ?v_{i-1,7}, ?v_{i-1,4}), ?v_{i-1,2}) \wedge
 - (10) ?v_{i,6} = {?v_{i,7} | ?v_{i,8}} \wedge ?v_{i,3} = {?v_{i,4} | ?v_{i,5}}]
-

Antecedents_{final}

- (11) FinalVector(?v_n, ?seq) \wedge ?state = Do(SetTblLoc(?v_{1,1}, ?v_{n,7}, ?v_{n,4}), ?v_{n,2}) \wedge
 - (12) Size(?v_{n,8}, ?n) \wedge ?size = (?settings + ?n)
-

Consequents

- (13) AchievableState(?state) \wedge SetTblLocs(?table, ?settings, ?state)

This rule extends sequences 1 \rightarrow N.

The term $?settings \leq ?size$ is produced because of the presence of a termination condition (second term of line 12) that is a mathematical constraint. This constraint contains two variables that are not effected by the *RIS* and a third known to never be negative (because it is the second argument to the predicate *Size*). For an acceptable non-negative $?n$ to be possible, it must be the case that the number of settings requested never exceed the capacity of the table selected.

Notice that the constraint that a sufficient number of utensils exist is not produced. This rule could waste much time attempting to set ten places on any of a hundred tables when only nine place settings exist. Improved reasoning about the implicit constraints in a sequential rule is an area for future research. This example is intended to motivate the need for doing so and illustrates a simple technique that demonstrates the feasibility of accomplishing this task

A related theme is that the issue of generalizing to N involves more than merely going from a fixed number of rule applications to an arbitrary number. Rather, it also involves the acquisition of concepts where there is a constrained *range* of acceptable numbers of rule applications.

Table 10.9 The Meaning of the Components in the RIS for Table Setting

<i>Component</i>	<i>Meaning</i>
1	the table
2	the current state
3	utensil sets available in the current state
4	the utensil set chosen at this step
5	the utensil sets left after this step
6	the clear locations on the table in the current state
7	the table location to set at this step
8	the table locations still clear after this step

AD-A191 327

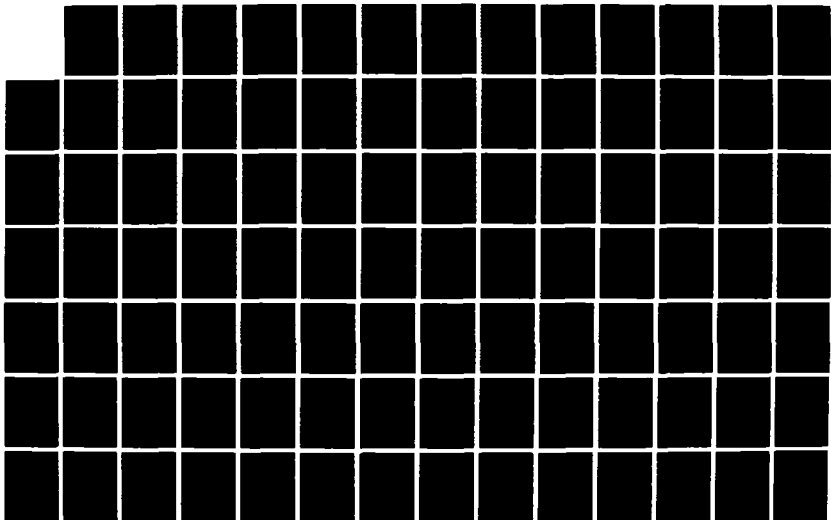
GENERALIZING THE STRUCTURE OF EXPLANATIONS IN
EXPLANATION-BASED LEARNING(U) ILLINOIS UNIV AT URBANA
COORDINATED SCIENCE LAB J H SHAYLIK DEC 87
UILLU-ENG-87-2276 N00014-86-K-0389

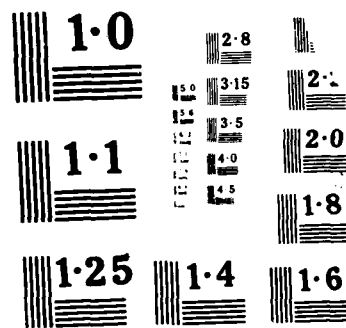
3/4

UNCLASSIFIED

F/O 23/2

NL





10.5. A General Version of DeMorgan's Law

The final example in this chapter illustrates the application of BAGGER to problems where situation calculus is not used. In these case, all terms are situation-independent and there is no need to unwind any rule applications to the initial state. Hence, in building the new sequential rule, terms are either omitted because they are axioms, dropped because they are satisfied by an earlier consequent, or included because they are situation-independent.

The proof of the construction on the left side of figure 8.6 appears in figure 10.14. This proof shows that a circuit design involving two connected *AND* gates can be implemented using *OR* and *NOT* gates. DeMorgan's law (for two-input gates) is used twice to prove this equivalence. The sequential rule that results is presented in table 10.10.

Table 10.11 demonstrates how this rule's *RIS* would be instantiated on a sample problem, one where a design for a four-input *AND* is requested. The *RIS* is instantiated from *N* to 1. At each step, one input to the *AND* gate is stripped off, until only two inputs remain. While this is happening, the inverted inputs are being combined into a cascaded collection of *OR* gates.

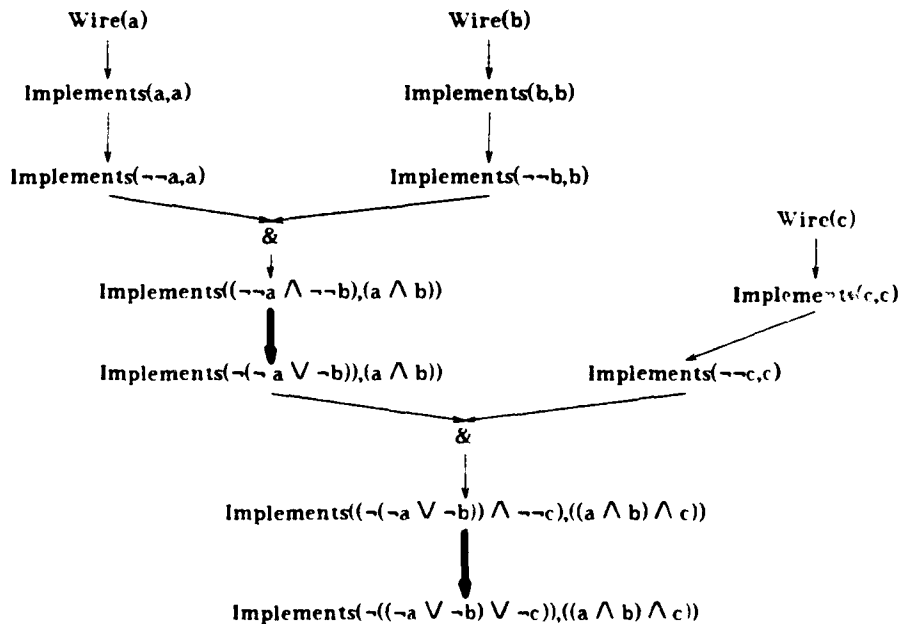


Figure 10.14 Implementing Two AND Gates with OR and NOT Gates

Table 10.10 A General Version of DeMorgan's Law

Antecedents _{initial}	
(1)	$\text{InitialVector}(\text{?v}_1, \text{?seq}) \wedge \text{?v}_{1,1} = \neg \text{?a} \wedge \text{Wire}(\text{?a}) \wedge \text{?v}_{1,3} = (\text{?a} \wedge \text{?v}_{1,4}) \wedge$
(2)	$\text{?v}_{1,2} = \neg \text{?v}_{1,4} \wedge \text{Wire}(\text{?v}_{1,4})$
Antecedent _{intermediate}	
(3)	$[\text{Member}(\text{?v}_i, \text{?seq}) \wedge \text{?v}_i \neq \text{?v}_1 \wedge \text{Member}(\text{?v}_{i-1}, \text{?seq}) \wedge \text{Predecessor}(\text{?v}_{i-1}, \text{?v}_i, \text{?seq})$ \rightarrow
(4)	$\text{?v}_{i,2} = \neg \text{?v}_{i,4} \wedge \text{Wire}(\text{?v}_{i,4}) \wedge \text{?v}_{i,3} = (\text{?v}_{i-1,3} \wedge \text{?v}_{i,4}) \wedge \text{?v}_{i,1} = (\text{?v}_{i-1,1} \vee \text{?v}_{i-1,2})]$
Antecedents _{final}	
(5)	$\text{Sequence}(\text{?seq}) \wedge \text{FinalVector}(\text{?v}_n, \text{?seq}) \wedge \text{?circuit} = \neg(\text{?v}_{n,1} \vee \text{?v}_{n,2}) \wedge \text{?spec} = \text{?v}_{n,3}$
Consequents	
(6)	$\text{Implements}(\text{?circuit}, \text{?spec})$
<i>This rule extends sequences $N \rightarrow 1$.</i>	

Table 10.11 Instantiating the RIS for the General DeMorgan's Law

Vector	$\text{?v}_{i,1}$	$\text{?v}_{i,2}$	$\text{?v}_{i,3}$	$\text{?v}_{i,4}$
3	$((\neg f \vee \neg g) \vee \neg h)$	$\neg i$	$((f \wedge g) \wedge h) \wedge i$	i
2	$(\neg f \vee \neg g)$	$\neg h$	$((f \wedge g) \wedge h)$	h
1	$\neg f$	$\neg g$	$(f \wedge g)$	g

In this example, **BAGGER** reformulated a solution in which a two-input version of DeMorgan's law is used twice in order to construct the equivalent of a three-input AND gate. The sequential rule acquired by the system represents a general version of DeMorgan's law. In

the acquired rule, all the inputs to an N -input *AND* gate are negated and used as the inputs to an N -input *OR* gate, whose output is passed through an inverter.

10.6. Summary

This chapter uses several examples, from a number of different domains, to illustrate the operation of the **BAGGER** generalization algorithm and to discuss its strengths and weaknesses. Topics discussed include how unwinding increases the operability but can restrict the generality of a learned rule (section 10.1), how the generality of a learned rule can be influenced by seemingly minor changes to a problem (section 10.2), and how the addition of extra terms in a sequential rule can increase its efficiency (section 10.4). Section 10.3 presents an example that **BAGGER** does not fully generalize and section 10.5 demonstrates the system's operation on examples not expressed in situation calculus. The next chapter contains a performance analysis of explanation-based learning in general and the **BAGGER** system in particular.

Chapter 11

Empirical Analysis of Explanation-Based Learning Algorithms

An empirical analysis of the performance of the **BAGGER** system is presented in this chapter. This system is compared to an implementation of a standard explanation-based generalization algorithm and to a problem-solving system that performs no learning. Two different training strategies are analyzed, along with several variants of the basic experiments. Information relevant to making decisions when designing an explanation-based learning system is reported. The results demonstrate the efficacy of generalizing to N in particular, and of explanation-based learning in general.

The two basic experiments compare systems that learn from their own problem solving to systems that learn by observing the actions of external agents. The external agent merely solves problems, there need not be any thought given to properly ordering the examples to facilitate learning. Hence, in this mode the learning systems can be viewed as *learning apprentices*

[Mitchell185], analyzing the normal actions of their users in order to absorb new knowledge. The experimental results indicate that substantially better performance gains can be achieved by observing the behavior of external agents. This occurs because the time spent internally solving complicated problems from first principles can dissipate much of the savings made by learning.

A number of other issues are analyzed. A second set of experiments investigates how the performance of the three problem solvers depends on the complexity of the space from which problems are selected. Next, details on the time spent learning, including how learning time grows with the size of explanations, are reported. Another set of experiments addresses the operability/generalizability trade-off. Following that, several strategies for determining the order to access acquired rules are considered. Next, the effect of resorting to standard explanation-based learning when BAGGER cannot generalize the structure of an explanation is analyzed. Finally, a second type of problem is investigated and more detailed experimental results are presented, before the results of the experiments are summarized.

11.1. Experimental Methodology

Experiments are run using blocks-world inference rules. An initial situation is created by generating ten blocks, each with a randomly-chosen width and height. One at a time, they are dropped from an arbitrary horizontal position over a table; if they fall in an unstable location, they are picked up and re-released over a new location. Once the ten blocks are placed¹, a randomly-chosen goal height is selected, centered above a second table. The goal height is determined by adding from one to four average block heights. In addition, the goal specifies a maximum height on towers. The difference between the minimum and maximum acceptable tower heights is equal to the maximum possible height of a block. The reason for this upper bound is explained later. A sample problem situation can be seen in figure 11.1. Some experiments involving the acquisition of plans for clearing objects are also described in this chapter. In these experiments the scene is set up as for tower-building, then an unclear block is randomly chosen as the block to be cleared.

¹ After the ten blocks are dropped to construct an initial situation, the database of assertions describing the initial state is constructed. The entries in this database are randomly ordered before the experiment begins.

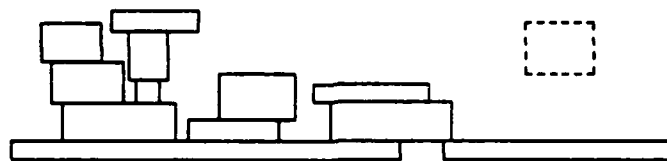


Figure 11.1 A Sample Problem

Once a scene is constructed, three different problem solvers attempt to satisfy the goal. The first is called **no-learn**, as it acquires no new rules during problem solving. The second, called **sEBL**, is an implementation of **EGGS** [Mooney86], a standard explanation-based generalization algorithm. **BAGGER** is the third system. All three of these systems use a backward-chaining problem solver to satisfy the preconditions of rules. **BAGGER**'s problem solver is extended as previously described in section 9.2. When the two learning systems attack a new problem, they first try to apply the rules they have acquired, possibly also using existing intra-situational rules. No inter-situational rules are used in combination with acquired rules, in order to limit searching, which would quickly become intractable. Hence, to be successful, an acquired rule must directly lead to a solution without using other inter-situational rules.

In **sEBL**, during generalization, explanation structures are pruned at terms that are either situation-independent or describe the initial state. Antecedents of new rules are grouped so that terms involving the same variables are located near each other, using the algorithm described for **BAGGER** in section 9.4.

Two different strategies for training the learning systems are employed. In one, called *autonomous mode*, the learning systems resort to solving a problem from "first principles" when none of their acquired rules can solve it. This means that the original inter-situational rules can be used, but learned rules are not used. When the proof of the solution to a problem is constructed in this manner, the systems apply their generalization algorithm and store any general rule that is produced. In the other strategy, called *training mode*, some number of solved problems (the *training set*) are initially presented to the systems, and the rules acquired from generalizing these solutions are applied to additional problems (the *test set*). Under this second strategy, if none of a system's acquired rules solves the problem at hand, the system is considered to have failed. No problem solving from first principles is ever performed by the learning systems in this mode.

Problem solving in the two modes is illustrated by figures 11.2 and 11.3, respectively. Notice that the acquired rules are not used when constructing new explanations. Partly this is a limitation due to computational resource restrictions, as explained in the next paragraph. However, it is also in the spirit of schema-based problem solving, where the idea is to rapidly

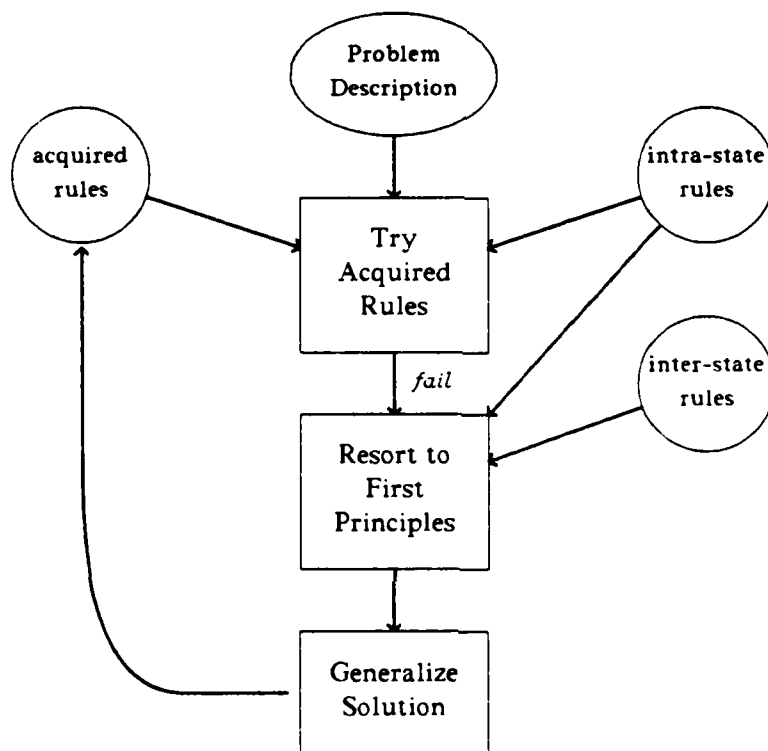


Figure 11.2 Problem Solving in the Autonomous Mode

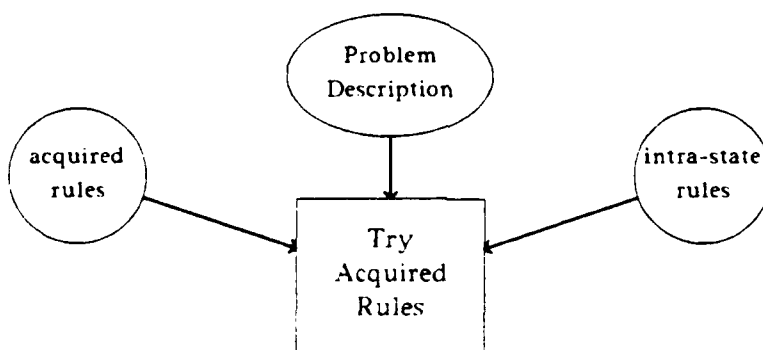


Figure 11.3 Problem Solving in the Training Mode

select and apply schemata, rather than spending large amounts of time connecting together many disjoint pieces of knowledge. While it might be reasonable to consider combining a handful of schemata, unrestricted combination can be too computationally expensive. In these experiments the limiting case of only allowing one schema to be applied is followed.

Unfortunately, constructing towers containing more than two blocks from first principles exceeds the limits of the computers used in the experiments. For this reason, the performance of the **no-learn** system is estimated by fitting an exponential curve to the data obtained from constructing towers of size one and two. This curve is used by all three systems to estimate the time needed to construct towers from first principles when required, and a specialized procedure is used to generate a solution. The estimated performance curve and the specialized algorithm are presented in sections 11.4 and 11.6, respectively.

Data collection in these experiments is accomplished as follows. Initially, the two learning systems possess no acquired rules. They are then exposed to a number of sample situations, building up their rule collections according to the learning strategy applied. (At each point, all three systems address the same randomly-generated problem. For each problem, the order the three problem solvers address it is randomly chosen.) Statistics are collected as the systems solve problems and learn. This continues for a fixed number of problems, constituting an experimental run. However, a single run can be greatly effected by the ordering of the sample problems. To provide better estimates of performance, multiple experimental runs are performed. At the start of each run, the rules acquired in the previous run are discarded. When completed, the results of all the runs are averaged together. The curves presented in this chapter are the result of superimposing 25 experimental runs and averaging.

Each learning system stores its acquired rules in a linear list. During problem solving, these rules are tried in order. The first successful one is used. When a rule is successful, it is moved to the front of the list. This way, less useful rules will migrate toward the back of the list. Analysis of other indexing strategies is presented in section 11.6.

This indexing strategy is the reason that, in the goal, tower heights are limited. The **sEBL** system would sooner or later encounter a goal requiring four blocks, and a rule for this would migrate to the front of its rule list. From that time on, regardless of the goal height, a four-block tower would be constructed. With a limit on tower heights, the rules for more efficiently

building towers of lower heights have an opportunity to be tried. This issue would be exacerbated if the goal was not limited to small towers due to simulation time restrictions.

These experiments were run on six identically configured Xerox Dandelion 1108's. Each machine has 3.5 megabytes of memory and runs the Koto release of Interlisp. Random numbers for the experiments are generated using the algorithm *RAN2* described on page 197 of [Press86]. The Interlisp function *RAND* is used to generate the seed. Timing is performed using the Interlisp function (*CLOCK 2*), which does not include garbage collection time. For efficiency reasons, the backward-chaining problem solvers use streams and generators [Charniak80].

Block dimensions are generated using a uniform probability distribution. The average block width is 75, while the average height is 35. The location blocks are dropped is uniformly distributed over a table 450 units wide. A decaying exponential distribution is used to generate tower heights. Scaling is such that the likelihood of a one-block tower is about twice that of a four-block tower.

Unfortunately these experiments measure some quantities that grow exponentially with problem size, which leads to large variances in the results. The fact that averages are produced from a small number of experimental runs due to computational resource limitations further increases the variance. The large variances should be taken into consideration when interpreting the results presented in this chapter. Appendix B presents tables containing the numbers, along with their standard deviations, used to plot most of the curves and histograms appearing in this chapter.

11.2. Comparison of the Two Training Strategies

This section is the first of several that present experimental results. In it the operation of the two basic modes of operation — autonomous and training — are analyzed and compared. Subsequent sections analyze variations of these experiments. Rather than comparing all of the variations to one another, the two basic experiments presented in this section serve as a baseline and later experiments are compared to them.

The autonomous mode is considered first. In this mode, whenever a system's current collection of acquired rules fails to solve a problem, a solution from first principles is constructed and generalized. Figure 11.4 shows the probability that the learning systems will

need to resort to first principles as a function of the number of sample problems experienced. As more problems are experienced, this probability decreases. (On the first problem the probability is always 1.) **BAGGER** is less likely to need to resort to first principles than is **sEBL** because **BAGGER** produces a more general rule by analyzing the solution to the first problem.

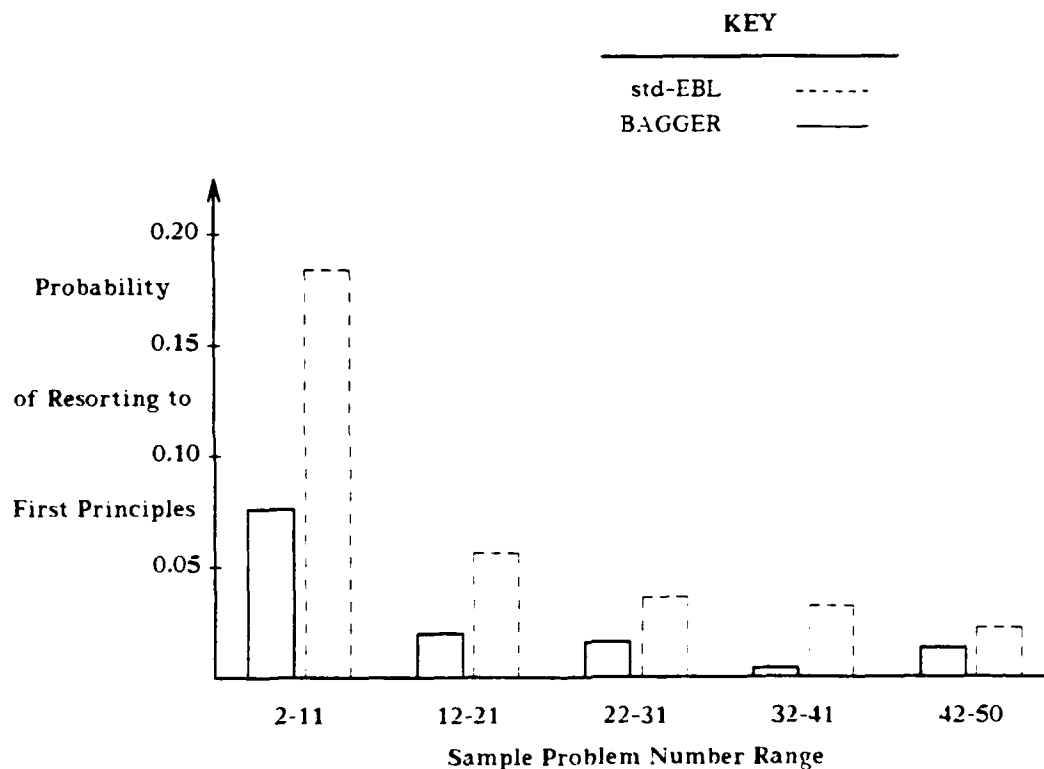


Figure 11.4 Probability of Resorting to First Principles in Autonomous Mode

On average, **BAGGER** learns 1.72 sequential-rules in each experimental run, while **sEBL** learns 4.28 rules. It takes **BAGGER** about 50 seconds and **sEBL** about 45 seconds to generalize a specific problem's solution. Averaging over problems 26—50 in each run (to estimate the asymptotic behavior), produces a mean solution time of 3720 seconds for **BAGGER**, 8100 seconds for **sEBL**, and 79,300 seconds² for **no-learn**. For **BAGGER**, this is a speed-up of 2.2 over **sEBL** and 21.3 over **no-learn**, where speed-up is defined as follows:

$$\text{Speed-up of } A \text{ over } B = \frac{\text{mean solution time for } B}{\text{mean solution time for } A}$$

Table 11.1 compares the speed of the three problem solvers over 625 sample problems (25 sample runs times the last 25 problems of each run). Recall that in each run, the three problem solvers all address the same problem at each point. The relative speeds of each are recorded and the table reflects how many times each system is the fastest, second fastest, and the slowest. Hence, **no-learn** solves about 20% of the problems faster than the two learning systems and about 60% of the problems slower than the other two. **BAGGER** solves slightly more than half the problems faster than do the other two systems. Only comparing the two learning systems, **BAGGER** solves about 70% of the problems faster than **sEBL** does.

Table 11.1 Relative Speed Summary in Autonomous Mode

	1st	2nd	3rd
<i>No-Learn</i>	20.2%	22.9	57.0
<i>Std-EBL</i>	24.8	41.6	33.6
<i>BAGGER</i>	55.0	35.5	9.4

BAGGER beats standard **EBL** 71.8%

² One day contains 86,400 seconds.

The numbers in this table only record the order of the three systems, they do not reflect by how much one system beats another. For example, building towers containing one block is often slightly faster to do from first principles, however towers of multiple blocks can be constructed much more rapidly by the learning systems. It takes **no-learn** about 10 seconds to build a tower with one block and 5×10^5 seconds for a tower of four blocks. For **BAGGER**, these averages are about 20 seconds and 70 seconds, respectively, for problems solved by its acquired rules. The performance separation seen in figure 11.4 is due to the fact that, when averaging numbers that vary by several orders of magnitude, the largest numbers heavily dominate.

Figure 11.5 plots the number of rules acquired as a function of problems experienced. The fact that the slopes of these curves are continually decreasing indicates that the time between learning episodes lengthens, which corresponds to the results of figure 11.4. That is, the mean time between failures of the acquired rules grows as more problems are experienced.

Figure 11.6 presents the performance during a single experimental run of the two learning systems in the autonomous mode. The average time to solve a problem is plotted, on a logarithmic scale, against the number of sample problems experienced. Notice that the time taken to produce a solution from first principles dominates the time taken to apply the acquired rules, accounting for the peaks in the curves.

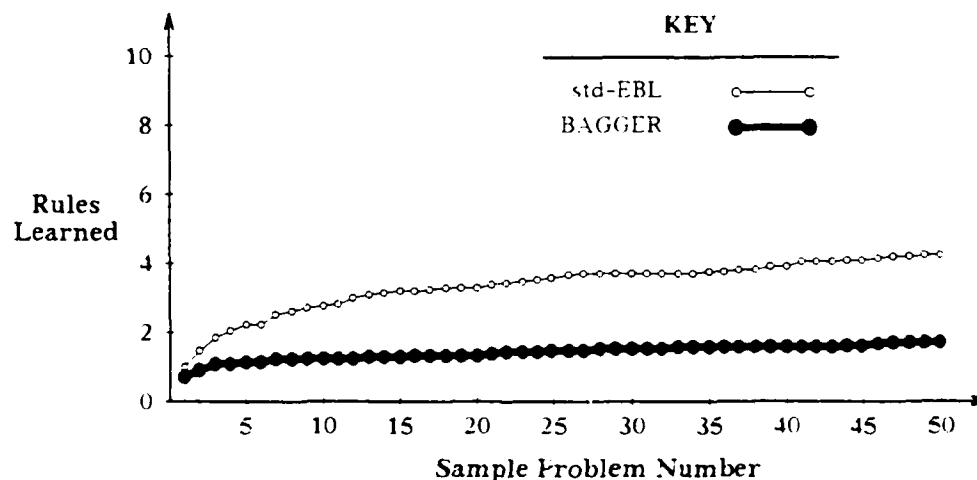


Figure 11.5 Rule Acquisition Comparison of the Autonomous Problem Solvers

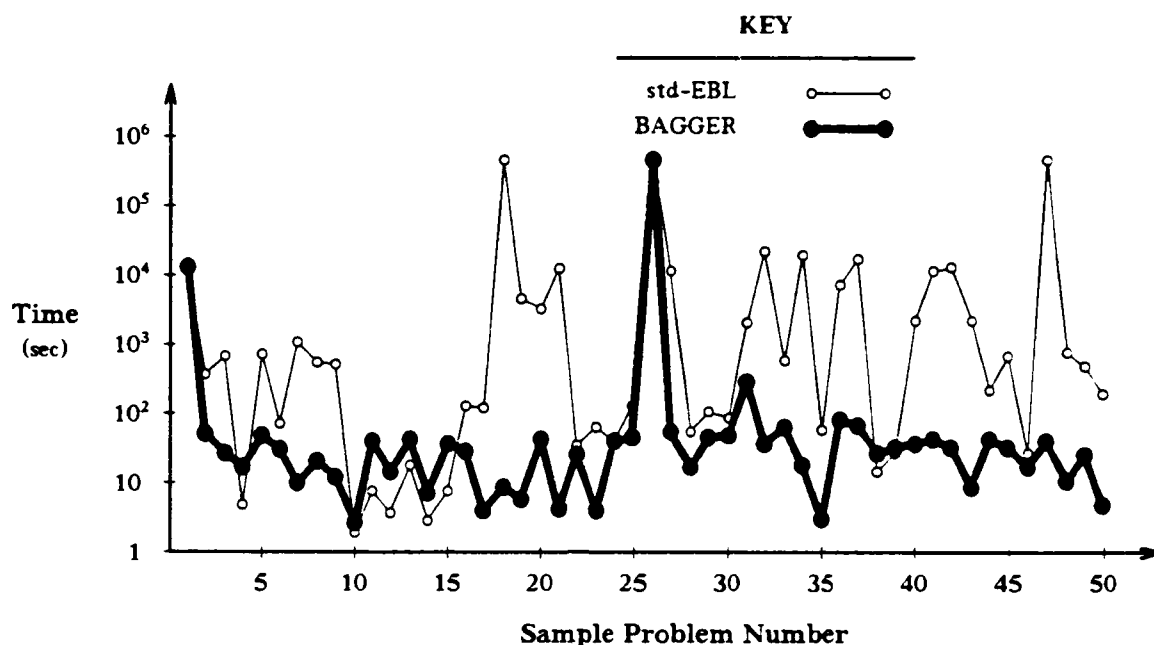


Figure 11.6 Performance Comparison of the Autonomous Problem Solvers

Because the cost of solving a big problem from first principles greatly dominates the cost of applying acquired rules, the autonomous mode may not be an acceptable strategy. Although learning in this mode means many problems will be solved quicker than without learning, the time occasionally taken to construct a solution when a system's acquired rules fail can dominate the performance. The peaks in the right-side of figure 11.6 illustrate this. A long period may be required before a learning system acquires enough rules to cover all future problem-solving episodes without resorting to first principles.

The second learning mode provides an alternative. If an expert is available to provide solutions to sample problems and an occasional failure to solve a problem is acceptable, this mode is attractive. Here, a number of sample solutions are provided and the learning systems generalize these solutions, discarding new rules that are variants³ of others already acquired. After training, the systems use their acquired rules to solve new problems. No problem solving

³ The algorithm for detecting variants determines if two rules exactly match, given some renaming of variables. This means, for instance, that $a \wedge b$ and $b \wedge a$ are not variants. Hence, semantically equivalent rules are not always considered variants. A more sophisticated variant algorithm would reduce the number of saved rules. However if the variant algorithm considered associativity and commutativity, it would be much less efficient [Benanav85].

from first principles is performed when a solution cannot be found using a system's acquired rules.

The performance results in the training mode are shown in figure 11.7. After ten training problems, the systems solve 20 additional problems. In these 20 test problems, the two learning systems never resort to using first principles. **BAGGER** takes, on average, 36.6 seconds on the test problems (versus 3720 seconds in the autonomous mode), **sEBL** requires an average of 828 seconds (versus 8100 seconds), and **no-learn** averages 68,400 seconds (versus 79,300 seconds).

Since **no-learn** operates the same in the two modes, these statistics indicate the random draw of problems produced an easier set in the second experiment. The substantial savings for the two learning systems (99% for **BAGGER** and 90% for **sEBL**) are due to the fact that in this mode these systems spend no time generating solutions from first principles. In this experiment, **BAGGER** has a speed-up of 22.6 over **sEBL** (versus 2.2 in the other experiment) and 1870 over **no-learn** (versus 23).

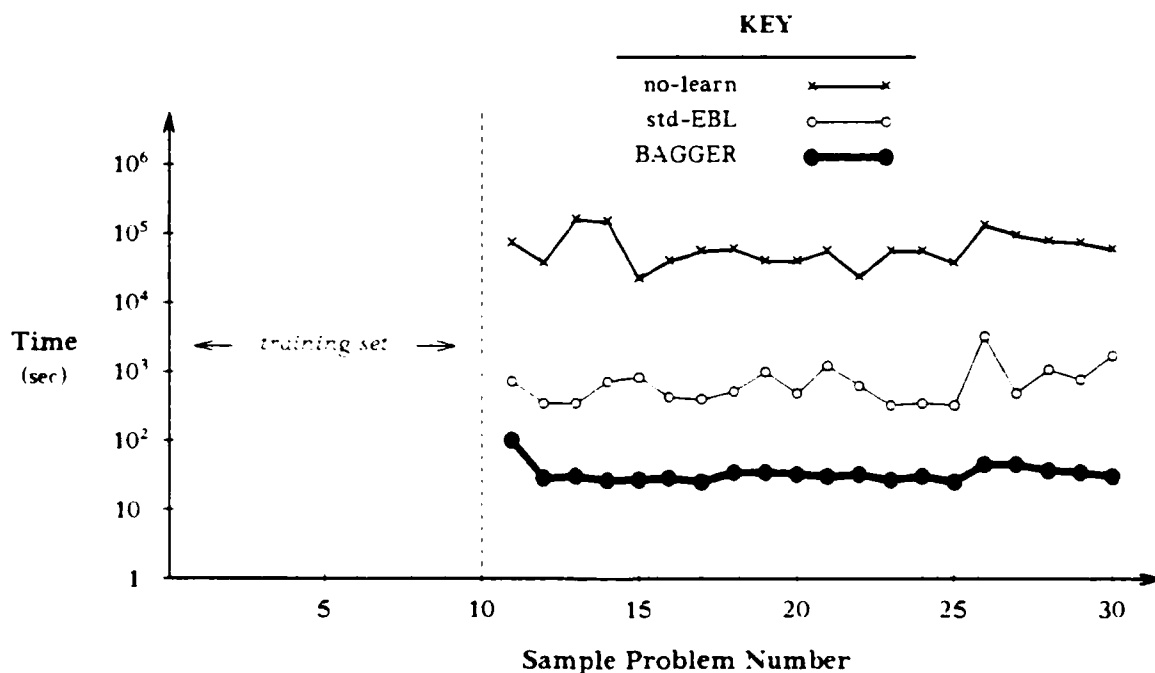


Figure 11.7 Performance Comparison of the Trained Problem Solvers

The relative speeds of the three systems in the training mode appear in table 11.2. (Only statistics from problems where all three problem solvers are successful are used to compute this table. As described later, this is about 98% of the test problems.) These numbers are comparable with the corresponding table for the autonomous mode. The main difference is that **sEBL** performs worse relative to the other systems (although its absolute performance is about ten times better than in the autonomous mode). The probable reason for this is that in the training mode the learning systems acquire more rules than in the autonomous mode.

The number of rules learned as a function of the size of the training set is plotted in figure 11.8. As before, **BAGGER** learns less rules than does **sEBL** and it approaches its asymptote

Table 11.2. Relative Speed Summary in Training Mode

	1st	2nd	3rd
<i>No-Learn</i>	20.8%	15.1	64.1
<i>Std-EBL</i>	21.6	47.8	30.7
<i>BAGGER</i>	57.6	37.1	5.2

BAGGER beats standard EBL: 75.4%

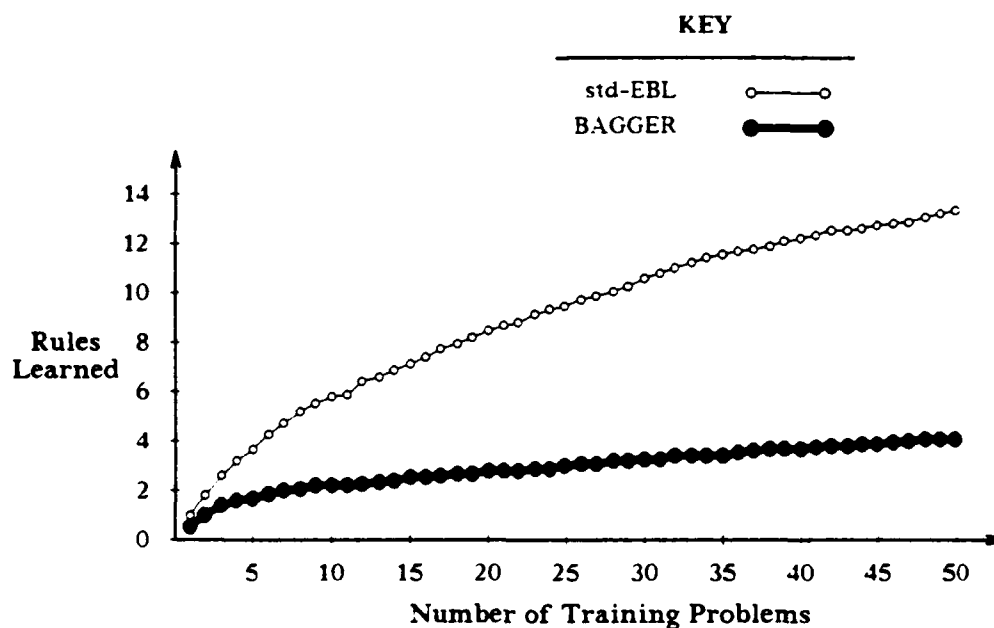


Figure 11.8 Rule Acquisition Comparison of the Trained Problem Solvers

sooner. Once the training set size exceeds about a half dozen examples, more rules are learned in the training mode than from 50 problems in the autonomous mode. This occurs because, during the training phase, new rules can be learned from problems that some previously-learned rule could have solved, albeit in a different way than the expert's solution. Recall that in the training mode the expert's solution is immediately given — the systems do not first try to solve the problem. Only general rules that are simple syntactic variants of previously-acquired rules are discarded.

One of the costs of using the training mode is that occasionally the learning systems will not be able to solve a problem. Figure 11.9 plots the number of failures as a function of the size of the training set. In each experimental run used to construct this figure, 20 test problems are solved after the training examples are presented. With ten training solutions, both of the systems solve over 98.5% of the test problems.

The final figure in this section, figure 11.10, summarizes the performance of the three systems in the two training modes. Note that a logarithmic scale is used. Both of the experiments demonstrate the value of explanation-based learning and also show the advantages of the **BAGGER** system over standard explanation-based generalization algorithms. **BAGGER** solves most problems faster than do the other two systems, its overall performance is better, and it learns less rules than does **sEBL**. Comparing the two training modes demonstrates the value of external guidance to learning systems. If a system solves all of its problems on its

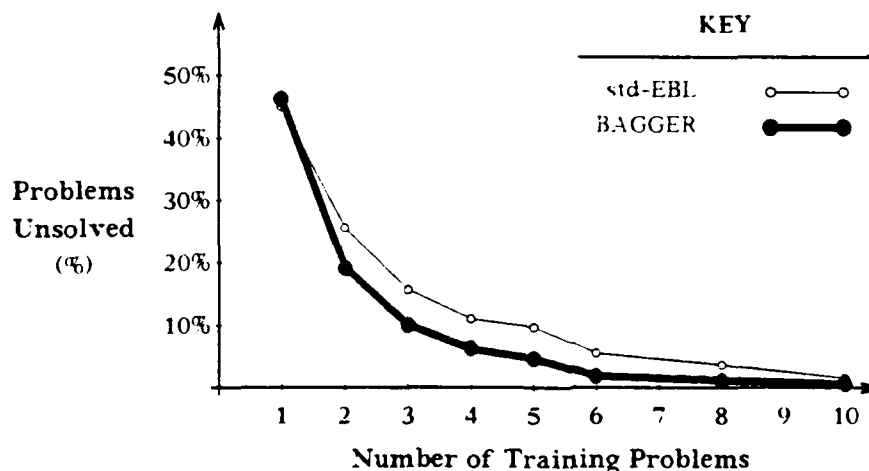


Figure 11.9 Failure Comparison of the Trained Problem Solvers

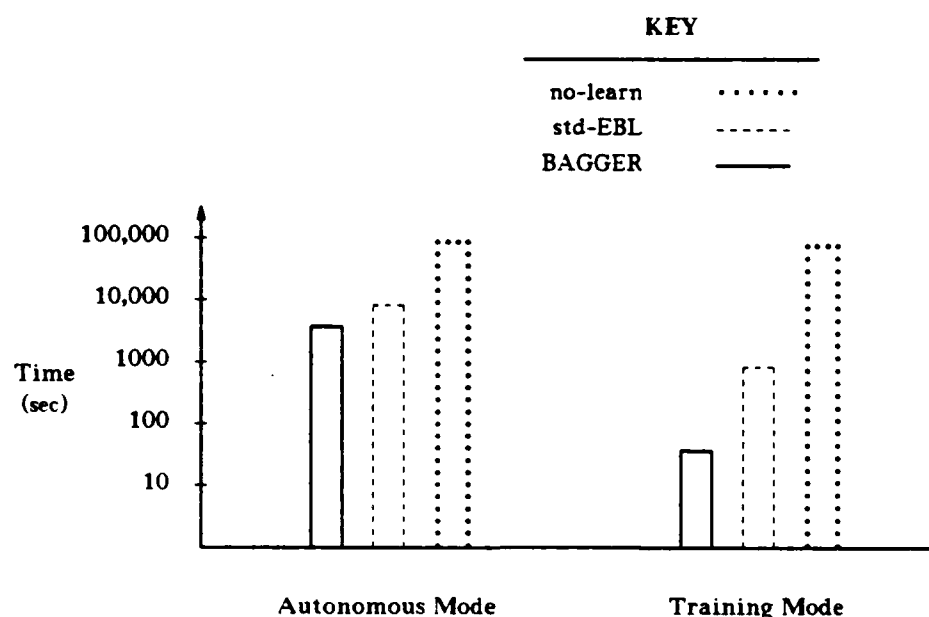


Figure 11.10 Performance of the Three Systems in the Two Modes

own, the cost of occasionally solving complicated problems from first principles can dissipate much of the gains from learning. The remainder of this chapter investigates variants on these experiments, comparing the results to the data reported in this section.

11.3. Effect of Increased Problem Complexity

The results in the previous section describe the behavior of the three systems for a class of problems, namely, the construction of towers containing from one to four blocks. An interesting question is how the relative performance of the three systems depends on the variability of the problems addressed. This section investigates performance as a function of problem complexity. Experiments are run on successively more complicated problem spaces, where complexity is varied by increasing the range of goal heights. Goals specify heights calculated by summing from one to N average blocks. The computational resources available restrict the maximum N to five. Since in each case, the lower bound on tower heights is one block, successive problem spaces subsume earlier ones.

One of the negative effects of learning new rules is that problem-solving time can often be wasted trying to apply them [Minton85, Mooney88]. This occurs when the preconditions of a new rule are checked but no successful binding of the rule's variables can be found. The

experiments in this section address this issue, because all of the acquired rules build towers but often no binding of a rule's preconditions can construct a tower of a given height. For example, there may be no way to bind the variables in an **sEBL** rule for building four-block towers such that the resulting tower is as tall as one average block. However, substantial time can be wasted discovering this. This negative effect also occurs for **BAGGER** rules. If **BAGGER** only learns that originally clear blocks can be moved to construct a tower, substantial time may be wasted before it finds out that there are not enough clear blocks in the initial state to achieve the specified height.

The figures in this section show how the performance of the three problem solvers depends on the magnitude of the range of possible goal heights. Figure 11.11 plots mean solution time, under both training modes, as a function of problem complexity. Since **BAGGER** learns no rules when the goal only involves moving one block, points are not plotted for the case when the maximum number of blocks in a tower is one. In all cases, the learning systems outperform the problem solver that does not learn. There is no evidence that, on average, learning degrades performance.

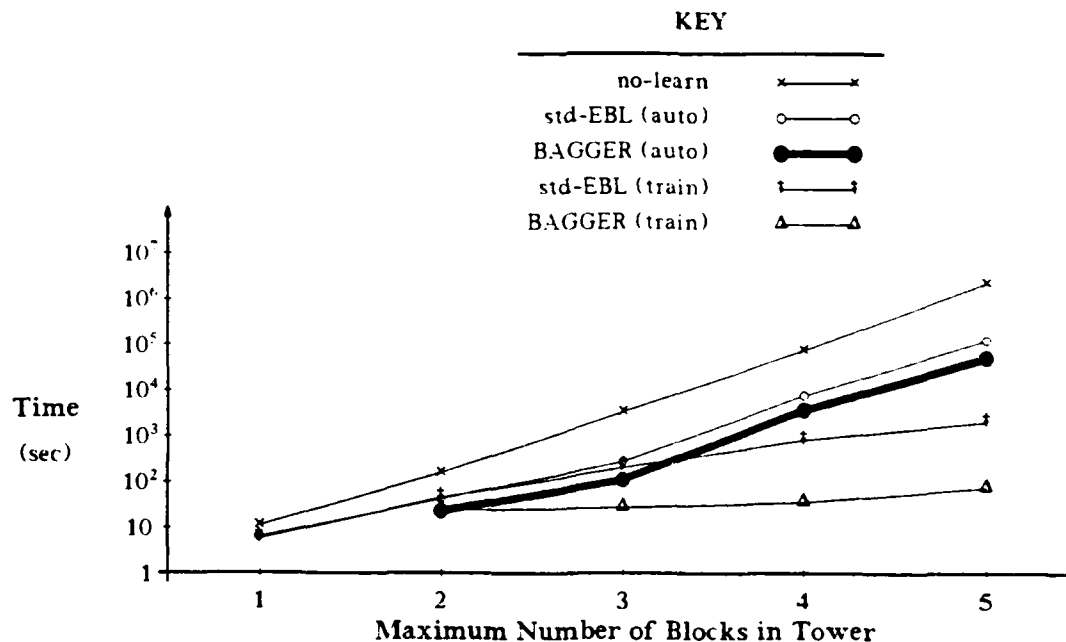


Figure 11.11 Mean Solution Time as a Function of Problem Complexity

Table 11.3 reports the various speed-ups produced by comparing the problem solvers for this collection of experimental runs. This data demonstrates that the performance difference among the various system configurations generally widens as the complexity of the problem space increases. The trend is not monotonic in the autonomous mode, possibly due to the disproportionate effect of the infrequent need to construct a solution from first principles. Since resorting to first principles occurs infrequently, statistical fluctuations strongly effect the speed-up ratios. The advantage of the training mode over the autonomous mode becomes more pronounced as problem complexity increases, because as larger towers are called for, the cost of building a solution from first principles becomes more dominant.

Figure 11.12 reports the percentage of problems where **no-learn** outperforms both of the learning systems. Even when the goal height only ranges from one to two blocks, **BAGGER** outperforms **sEBL** on over half of the problems. As the complexity of the problem space increases, **BAGGER** increasingly outperforms **sEBL**.

Table 11.3 Speed-Up as a Function of Problem Complexity			
<i>Tower Size Range</i>	<i>BAGGER over sEBL</i>	<i>BAGGER over No-Learn</i>	<i>sEBL over No-Learn</i>
autonomous mode			
1-1	—	—	1.73
1-2	1.81	7.01	3.87
1-3	2.75	30.20	11.00
1-4	2.18	21.30	9.79
1-5	2.25	39.00	17.30
training mode			
1-1	—	—	1.97
1-2	1.92	6.35	3.32
1-3	6.69	110.00	16.40
1-4	22.60	1,870.00	82.60
1-5	27.10	24,100.00	889.00

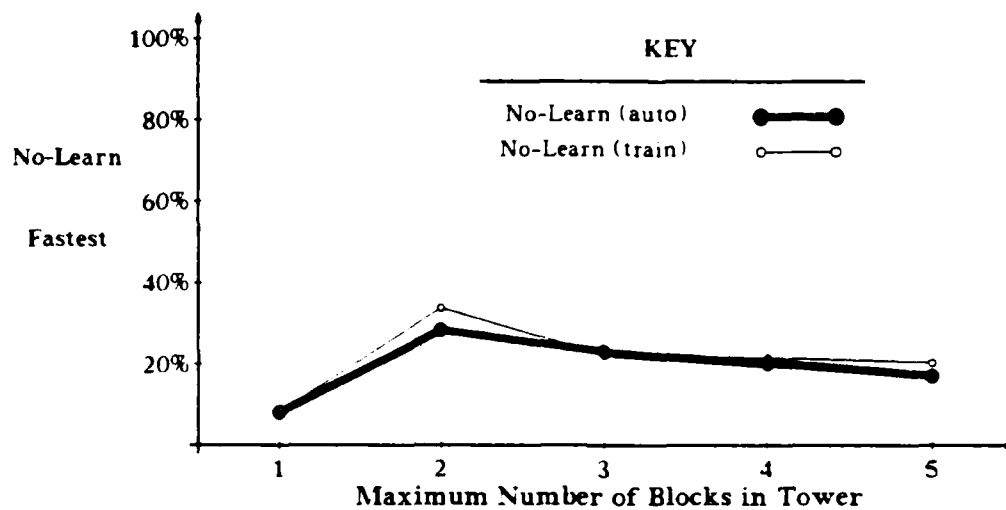


Figure 11.12 BAGGER Faster than sEBL as a Function of Problem Complexity

The number of rules learned under the various configurations is plotted in figure 11.13. The number of rules acquired grows roughly linearly with the complexity of the problem space.

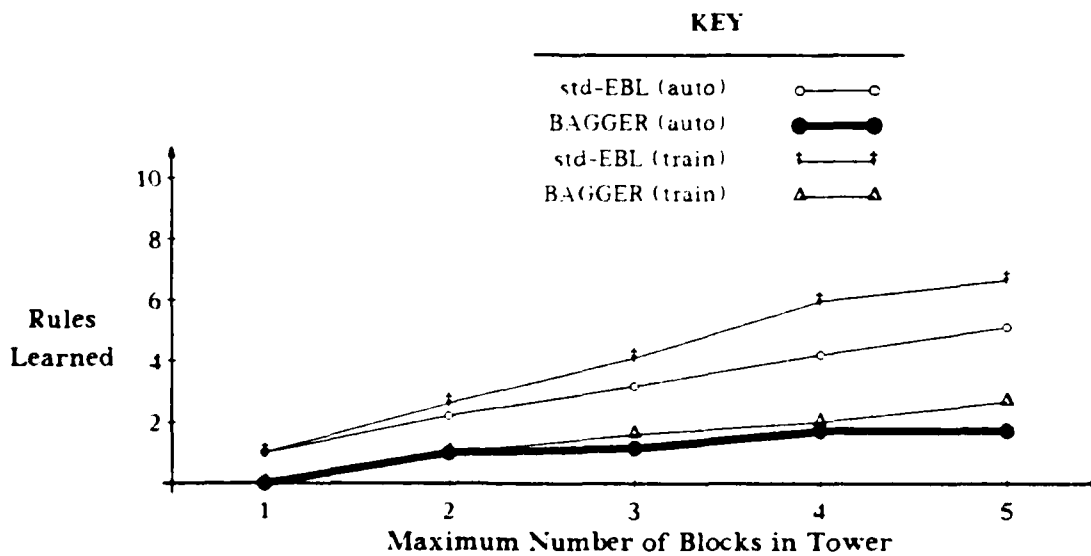


Figure 11.13 Rules Learned as a Function of Problem Complexity

Figure 11.14 plots the mean number of disjuncts in BAGGER's sequential rules as a function of problem complexity. As the average problem gets more complicated, the solutions get more complicated and, hence, BAGGER has more focus rules to analyze. This increases the likelihood that more ways of satisfying the preconditions of the focus rule will be encountered.

The experiments reported in this section demonstrate that the gains of explanation-based learning increase as the range of potential problems increases. They provide no evidence for the conjecture that learning can impede overall problem-solving performance. Rather, they indicate that as the range of problems increases, learning is increasingly beneficial. In addition, the relative performance of BAGGER over sEBL grows as the range of possible problems increases. Finally, these experiments strengthen the argument that the training mode is preferable to the autonomous mode of operation.

11.4. Time Spent Learning

This section investigates how much time is spent within the generalization algorithms. First this is viewed from the context of the operation of the two basic modes of operation. Following that, the time required to generalize an explanation is reported as a function of the size of the explanation.

Figures 11.15 and 11.16 plot how much time is spent by the generalization algorithms in the two modes of operation. In autonomous mode, successively less time is spent learning because as more problems are experienced it is less likely that learning will be required.

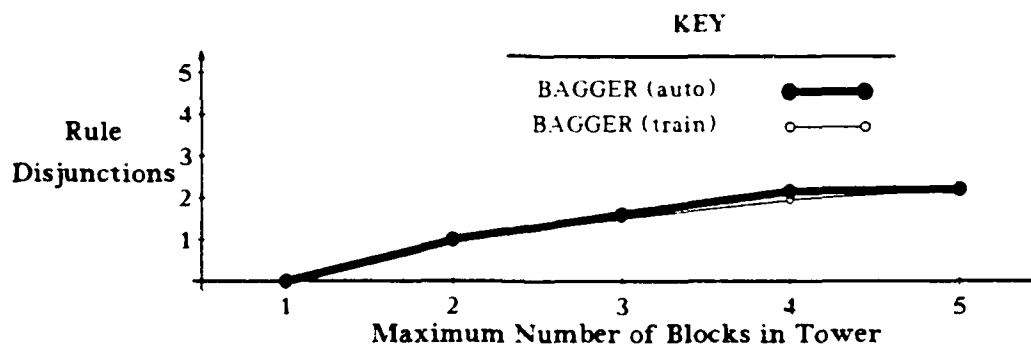


Figure 11.14 BAGGER Rule Disjunctions as a Function of Problem Complexity

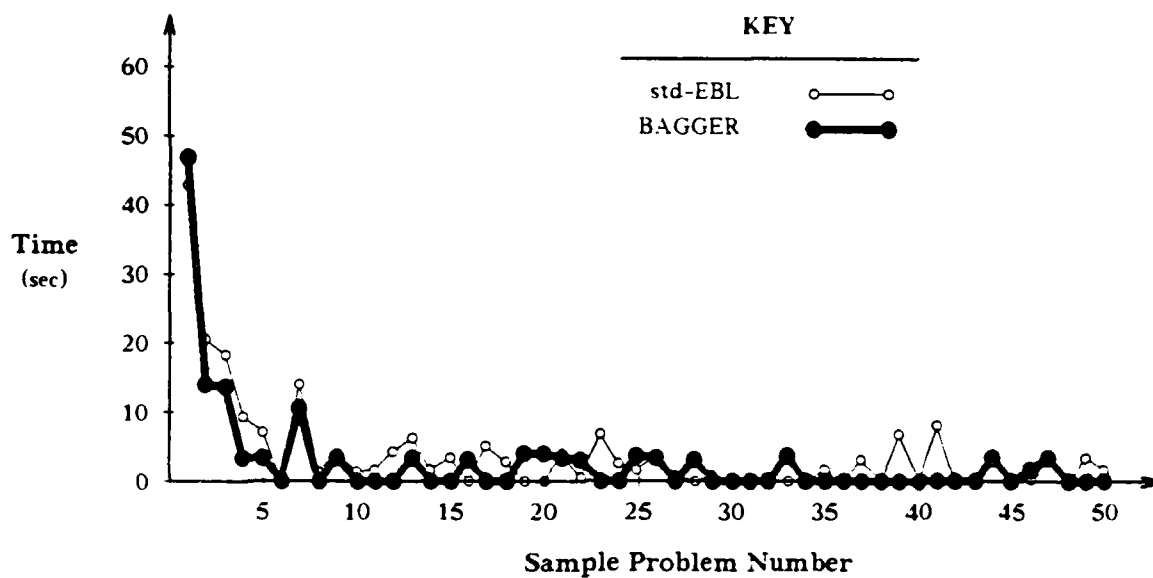


Figure 11.15 Mean Time Spent Learning in Autonomous Mode

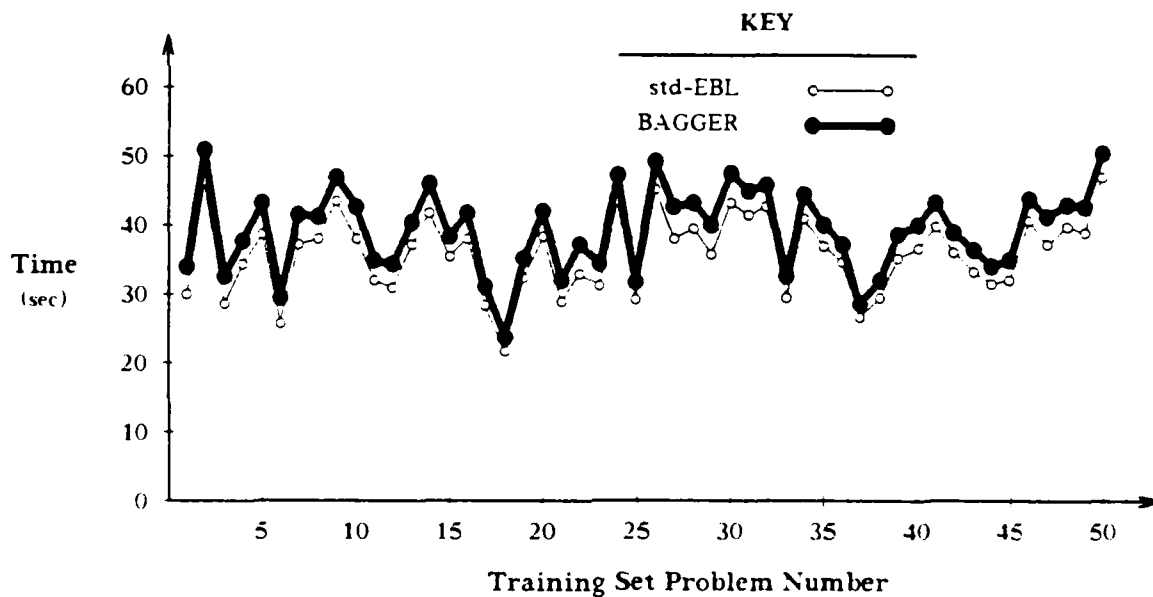


Figure 11.16 Mean Time Spent Learning in Training Mode

Conversely in training mode, there is a slight increase in time spent learning because the learners must check if a new rule is a variant of any of the previously acquired ones.

The next three figures describe how the learning time depends on the size of explanations. Figure 11.17 presents the time spent learning, in the training mode, as a function of the number of blocks moved. Although more complicated than standard explanation-based generalization algorithms, BAGGER takes only slightly more time to generalize an explanation.

The time spent learning in the training mode can be divided into three components.

$$\begin{aligned}
 \text{total time spent learning} &= \text{time to build explanation from user's actions} + \\
 &\quad \text{time to generalize explanation} + \\
 &\quad \text{time to reorganize and save new rule}
 \end{aligned}$$

The learning time curves in figure 11.17 reflect this decomposition. The time to reorganize and save a new rule includes looking to see if it is a variant of an old rule. In the training mode, the externally-provided solutions only indicate which blocks are moved and where they are placed. The systems must validate these steps and ascertain they achieve the desired goal.

A theoretical analysis of the problem-solving speed-up that can be achieved by providing steps within a proof appears in chapter 4 of [Mooney88]. His results indicate an exponential speed-up is possible, although problem solving still is exponential. Comparing the results in figure 11.17 concerning explanation construction to the estimated performance of **no-learn** (section 11.9) produces a speed-up of $19^N - 1$, where N is the number of blocks moved.

Figure 11.18 shows that the size of explanations grows exponentially with the number of blocks moved. The size of both the full explanation and of the pruned explanation is reported. About one-third of the nodes are pruned.

How the time spent learning depends on the size of the explanation appears in figure 11.19. This figure reports how much time is spent, per node, explaining the user's actions and then generalizing the explanations. The ratio for explanation construction uses the size of the full

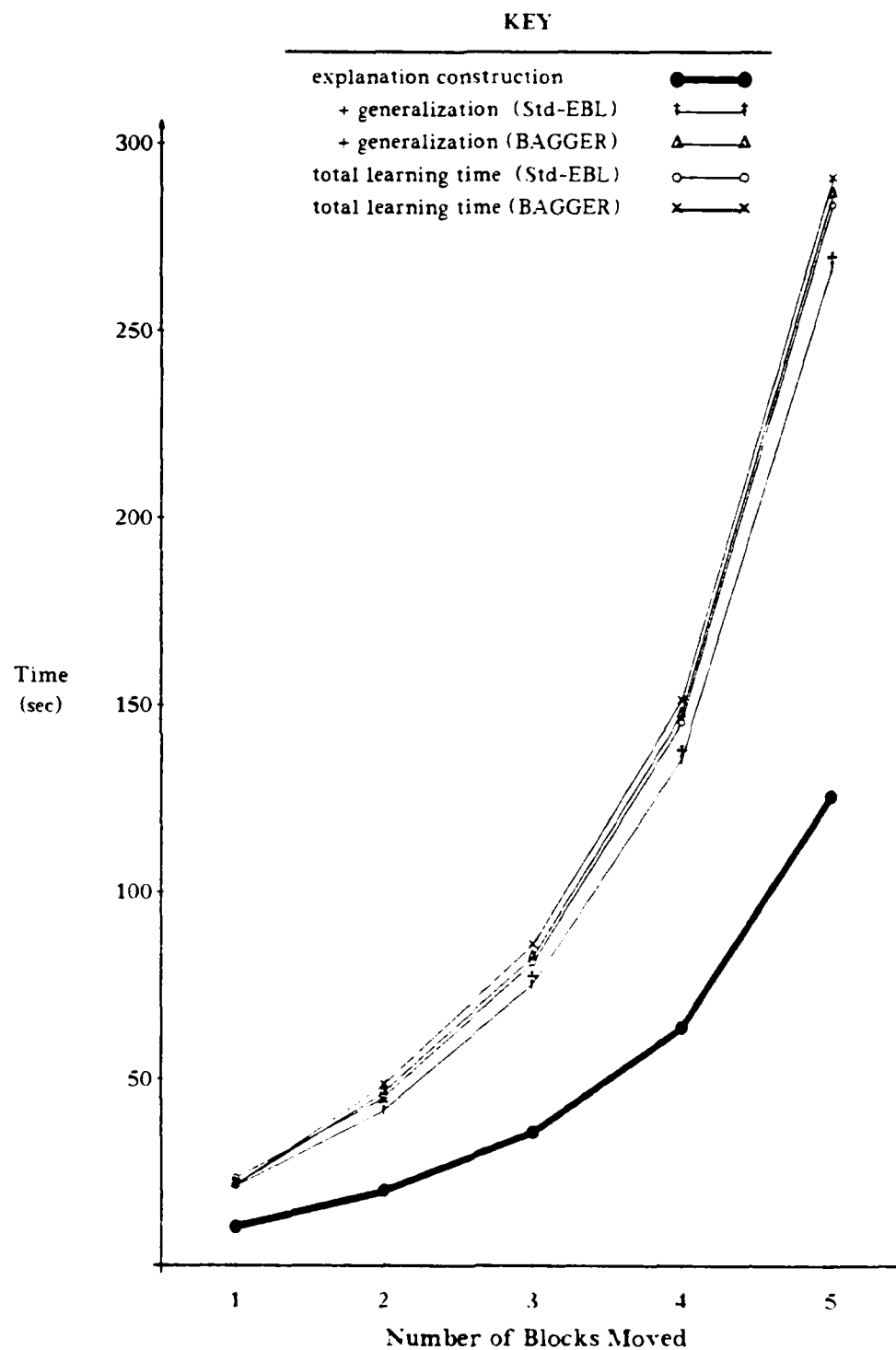


Figure 11.17 Time Spent Learning as a Function of Blocks Moved

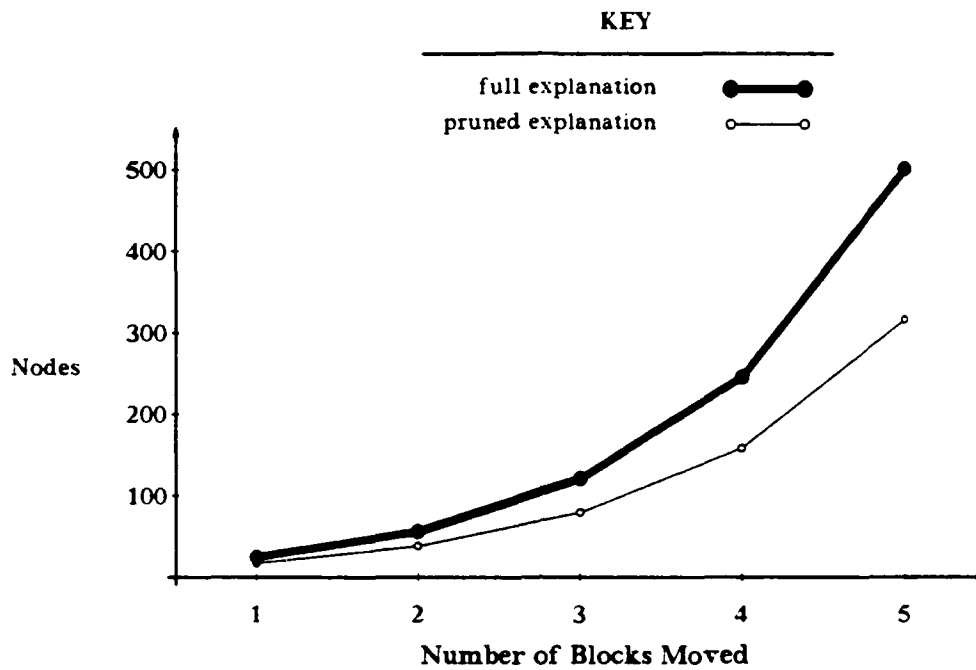


Figure 11.18 Explanation Size as a Function of Blocks Moved

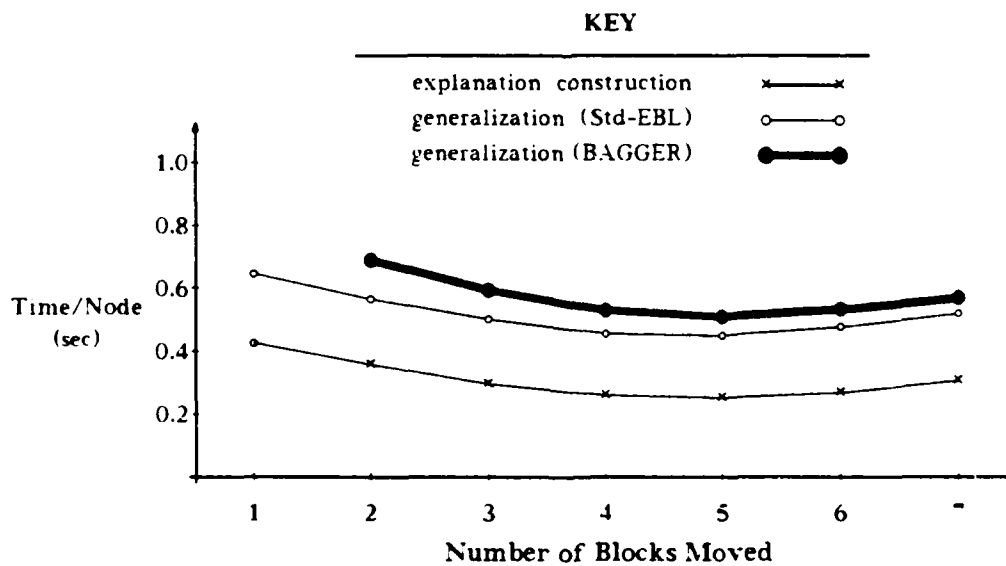


Figure 11.19 Time Spent per Explanation Node

explanation, while the pruned explanation is used for the generalization time ratios. In addition, the generalization time ratios only include the time spent generalizing the pruned explanation; the time spent reorganizing and saving the rule is *not* included. No point is plotted for **BAGGER** when moving one block because in that case no focus rule is found in the explanation and, hence, no new rule is produced. All three curves have approximately the same shape and **BAGGER** spends only slightly more time than **sEBL**. Assuming a fixed overhead during generalization (to account for the initial declines), these curves indicate that empirically both **BAGGER** and **sEBL** grow more than linearly in terms of the number of nodes in the explanation. However, a proof that the **EGGS** algorithm (the basis of **sEBL**) could be performed in linear time appears in [Mooney88]. This proof is based on a linear unification algorithm [Paterson78], which is not used in **BAGGER** nor **sEBL** due to its large constant overhead.

11.5. Operationality/Generality Trade-Off

The previous chapter discusses how **BAGGER** can be instructed to construct more operational sequential rules. Basically, this entails not pruning any of the nodes in an explanation. This section describes the effect on both **BAGGER** and **sEBL** of not pruning nodes.

Performance in the autonomous mode under this condition appears in figures 11.20 and 11.21. When constructing more operational rules, the learning systems more frequently resort to solving problems from first principles. This occurs because the more operational rules, being less general, are more likely to fail.

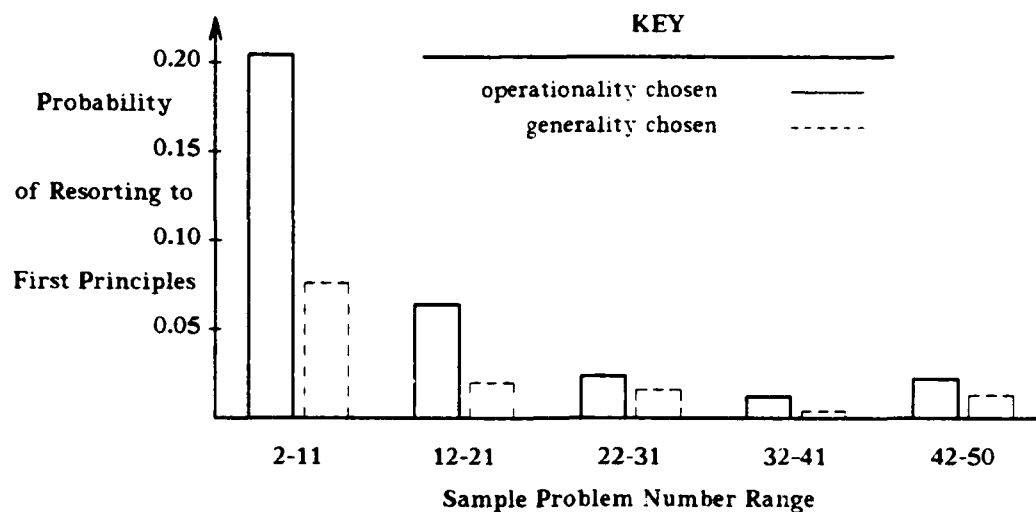


Figure 11.20 Operationality Results in Autonomous Mode for BAGGER

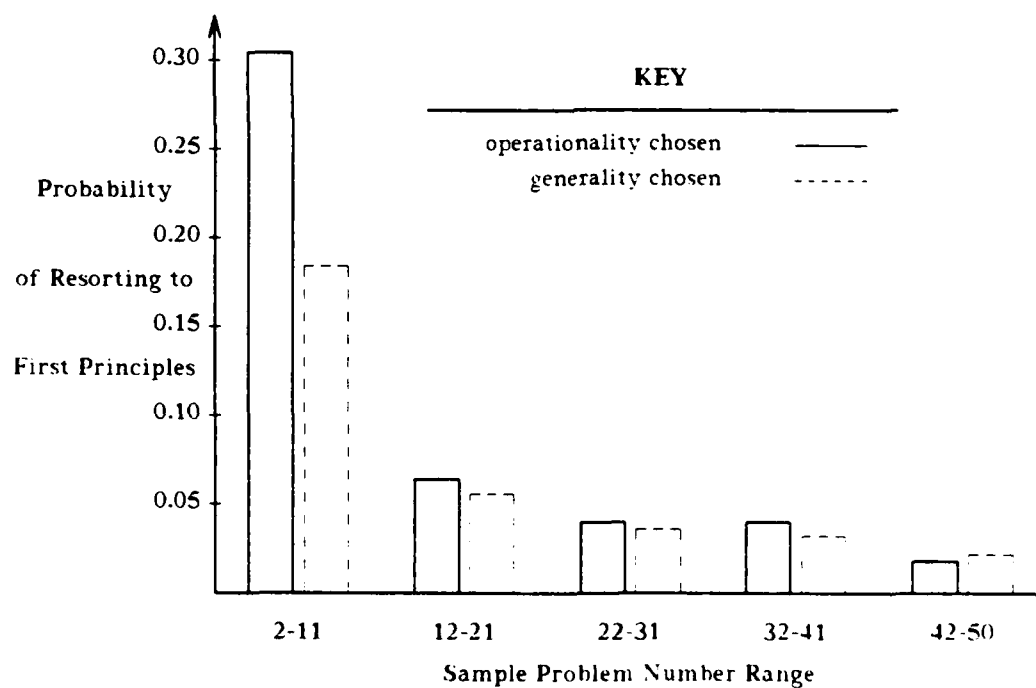


Figure 11.21 Operationality Results in Autonomous Mode for Standard EBL

Table 11.4 Relative Speed Summary for Operational Version (Autonomous)

	1st	2nd	3rd
<i>No-Learn</i>	1.1% (20.2) ⁴	24.2 (22.9)	74.7 (57.0)
<i>Std-EBL</i>	29.5 (24.8)	47.0 (41.6)	23.5 (33.6)
<i>BAGGER</i>	69.4 (55.0)	28.8 (35.5)	1.8 (9.4)

BAGGER beats standard EBL: 70.2% (71.8)

The relative speeds of the problem solvers, in their operational versions in the autonomous mode, appears in table 11.4. In the operational version, **no-learn** performs substantially worse, relatively. Most of **no-learn**'s loss is transferred to **BAGGER**.

Figure 11.22 compares the operational and general versions in the training mode. The advantage of the operational version is clearly visible, especially for **sEBL**. Table 11.5 reports the relative speeds for this situation. Unlike in the autonomous mode, **no-learn**'s relative performance is only slightly degraded. This may be due to the fact that more rules are learned in the training mode, possibly increasing the likelihood that time is wasted on an unsuccessful rule or that a unnecessarily complicated, but usually successful, rule is at the front of the rule queue.

Table 11.6 summarizes the major differences between the operational and general versions. Results from the two cases are presented for various statistics, along with the ratio of the two results. The results indicate that, although more rules are learned and used, the fact that they are evaluated faster leads to quicker overall solutions. The one exception to this is **BAGGER** in autonomous mode. Here the need for more solutions from first principles eliminates the gain from the more rapid evaluation of the operation rules.

⁴ The numbers in parentheses in this and the next table are the corresponding results previously reported for the experiments involving the standard versions of the learners.

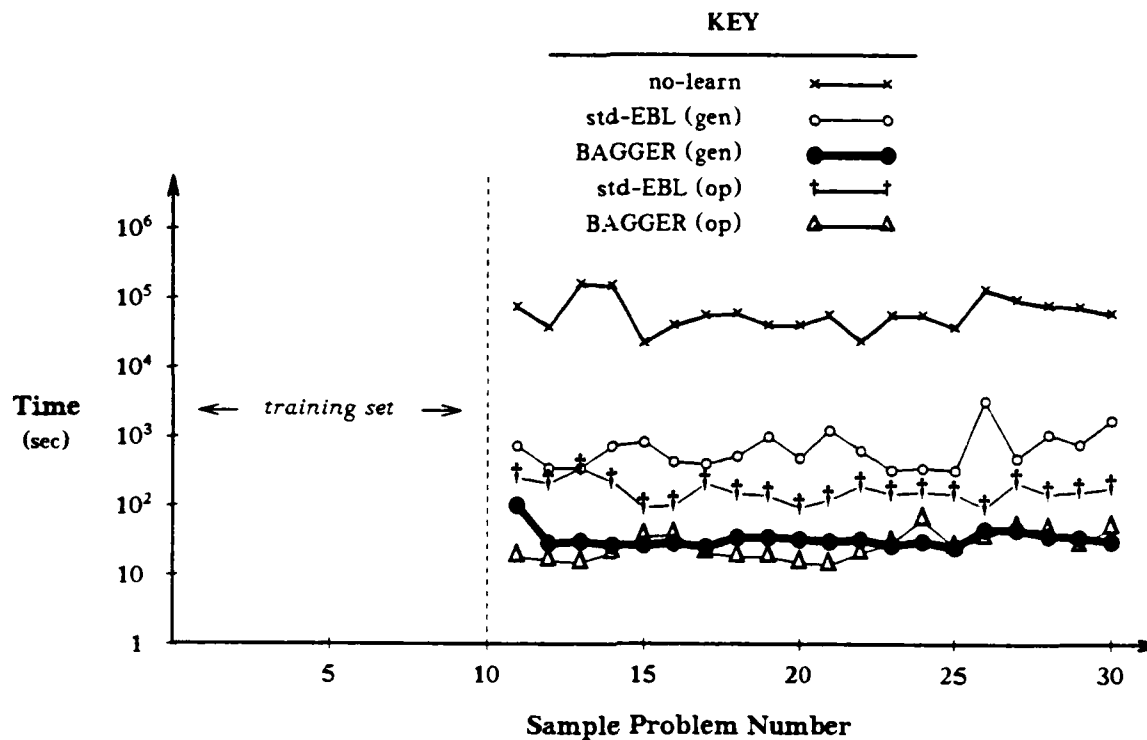


Figure 11.22 Operationality and Generality Results in Training Mode

Table 11.5 Relative Speed Summary for Operational Version (Trained)

	1st	2nd	3rd
<i>No-Learn</i>	17.6% (20.8)	15.0 (15.1)	67.4 (64.1)
<i>Std-EBL</i>	26.9 (21.6)	44.1 (47.8)	29.0 (30.7)
<i>BAGGER</i>	55.5 (57.6)	40.9 (37.1)	3.6 (5.2)

BAGGER beats standard EBL: 71.21% (75.4)

Table 11.6 Operationality vs. Generality Results			
<i>Description</i>	<i>Operationality Chosen</i>	<i>Generality Chosen</i>	<i>Ratio (Op/Gen.)</i>
Mean Solution Time			
autonomous mode			
<i>std-EBL</i>	6790 sec	8100	0.83
<i>BAGGER</i>	6660	3720	1.79
training mode			
<i>std-EBL</i>	155(174) ⁵	779(828)	0.20
<i>BAGGER</i>	27(29)	36(37)	0.75
Rules Learned			
autonomous mode			
<i>std-EBL</i>	5.52	4.28	1.29
<i>BAGGER</i>	2.60	1.72	1.51
training mode			
<i>std-EBL</i>	6.88	5.96	1.15
<i>BAGGER</i>	3.24	2.01	1.61
Probability of Successful Application			
autonomous mode			
<i>std-EBL</i>	0.50	0.56	0.89
<i>BAGGER</i>	0.90	0.99	0.91
training mode			
<i>std-EBL</i>	0.41	0.59	0.69
<i>BAGGER</i>	0.79	0.99	0.79
Percentage Solved			
training mode			
<i>std-EBL</i>	91.4%	98.5	0.93
<i>BAGGER</i>	94.0	99.4	0.95

⁵ Numbers in parentheses indicate mean solution time for *all* problems, including problems not solved.

The probability of a successful rule application is defined by:

$$Probability(success) = \frac{\sum_{rules} successful\ applications}{\sum_{rules} attempted\ applications} .$$

The average number of rules tried when solving a problem is inversely proportional to this number. Hence, the data in the table shows that more rules are evaluated, on average, when solving problems with the operational rules.

The final point in this section is that, in training mode, the probability that the test problem is not solved is higher in the operational version. This occurs because a higher number of operational rules are needed to cover the same number of future problems. Thus, for the operational version to be attractive, the training set must contain a highly representative sample of future problems.

11.6. Rule Access Strategies

The primary strategy used to organize acquired rules is to keep them in a linear list, moving a rule to the front of this list whenever it successfully solves a problem. During problem solving, the rules are tried in order. The ordering of the rules in this list can greatly effect problem-solving performance. Seven additional strategies for accessing acquired rules have also been investigated. After describing the eight strategies for selecting the order to try rules, their performance is compared.

This section largely investigates a question of problem solving, rather than one of learning. The question is how should the results of learning best be organized to increase the efficiency of problem solving. The access strategies investigated have no effect on the number of rules learned, nor on the probability of a successful solution. All that differs is the order the rules are checked. If necessary, all of them will be tried.

In all of the strategies, acquired rules are organized in a linear list. More complicated data structures, such as discrimination networks [Charniak80], can improve problem-solving performance. However, all of the rules organized in these experiments satisfy the same goal, namely, the construction of a tower. Since they all contain the same predicates in their consequent, it is reasonable to assume that they are all grouped at the same node in a

discrimination net. Hence, the results of these experiments are relevant to more complicated data structures, and even to parallel architectures, provided at some point a group of candidates are serially visited.

Strategy Descriptions

The eight strategies are described below and are presented roughly in order of complexity. These access strategies can be divided into two qualitative groups. The first four strategies are independent of problem-solving performance. With these, the complication of collecting statistics during problem solving is avoided. The second four strategies dynamically depend on the examples in the test set. That is, the results of problem solving continually effect the order in which rules are accessed.

Only the training mode is used in these experiments and the training set always contains ten problems, while the test set contains twenty. More rules are learned, on average, in this mode, making it a better vehicle for investigating the issue of rule access strategies. In all of the experimental runs, towers are constructed and the goal height is randomly determined by summing from one to four average block heights. With a training set of ten problems, the two learning systems both solve more than 98% of the test problems. Since problems not soluble by a given collection of rules are more likely to be anomalous, only the measurements on successful solutions are considered when collecting the data reported in this section.

Most Recently Learned (MRL)

In the first strategy, rules are accessed in *reverse* order from the order they are learned. This is easily implemented by pushing, during the training phase, new rules onto the front of the list of acquired rules. Unless otherwise stated, the other strategies use this method to initially insert new rules into the list of previously acquired rules.

Least Recently Learned (LRL)

In the second strategy, rules are accessed in the same order as they are learned. This is implemented by placing new rules at the *end* of the list of acquired rules.

Sorted by Situations Traversed (SORT)

This strategy applies when using situation calculus and is only relevant to rules learned by standard explanation-based learning (i.e., non number-generalized rules). By looking at

the consequent of a new rule, the number of situations traversed by applying the rule can be determined. Once the training phase completes, the rules are sorted so that those that traverse the fewest number of states are in the front of the list of acquired rules. This list is not altered during the test phase. (This strategy shares many of the characteristics of iterative-deepening [Korf85].) Under this strategy, BAGGER rules follows the *MRL* strategy.

Randomly Selected (RAND)

In this strategy, the order that rules are accessed is randomly determined. This is implemented by randomizing the list of acquired rules before *each* test problem, then trying to apply them in their randomized order.

Most Recently Used (MRU)

In the *MRU* strategy, which is the access strategy used elsewhere in this chapter, the order that rules are accessed depends on the last time they are successfully applied. This is implemented by, following a successful application, moving the successful rule to the front of the list of rules. The hypothesis of this strategy is that the more useful a rule is, the more likely it will be tried early.

This and the next three strategies can be viewed as if each rule is somehow scored and these scores are used to sort the rules so that highest scoring rules are at the front of the list. The implementations of the following three strategies use such a sorting strategy, although, as described in the above paragraph, *MRU* does not. For *MRU*, no statistics are associated with each rule. The expressions used for predicting a rule's future value under each strategy are listed after each is description.

$$Value_{MRU} = \text{time of last successful application}$$

Most Frequently Used (MFU)

In this strategy, the order that rules are accessed depends on how often they have been successfully applied. Unlike *MRU*, this strategy, as do the following two, requires additional information be recorded. In this case, each rule records the number of time it is used to solve a problem. *MFU* is implemented by, after each successful solution, sorting

the list of acquired rules so that the one with the highest number of successful applications is tried first.

$$Value_{MFL} = \text{number of successful applications}$$

Most Successfully Used (MSU)

This strategy orders rules according to the *a posteriori* probability they are useful. This probability is calculated by dividing the number of successful applications of the rule by the number of times the rule is tried. Rules never tried are ordered after rules successfully applied and before rules tried without success. This strategy requires two statistics be kept for each rule.

$$Value_{MSU} = \frac{\text{number of successful applications}}{\text{number of attempts to apply rule}}$$

Most Efficiently Used (MEU)

This strategy involves another way to get previously useful rules near the front of the list of acquired rules. Here the measure of a rule is determined by recording the total amount of time spent trying to satisfy the rule's preconditions and then dividing this time by the number of successful applications of the rule. This measures the time spent per successful application, and the lower this number the more promising the rule. Hence, this measurement is inversely proportional to the value of the rule. Rules never tried are ordered as in *MSU*. Again, two statistics must be kept.

$$Value_{MEU}^{-1} = \frac{\text{total time spent trying this rule}}{\text{number of successful applications}}$$

Results

Figure 11.23 presents, for each strategy for accessing rules, its performance on the problems in the training set. In this figure, the strategies are organized so that the most efficient one (for *SEBL*) is on the left and the least efficient one is on the right. There is about an 8-fold difference between the best and the worst. The speed-up of **BAGGER** over *SEBL* ranges from 5 to 40.

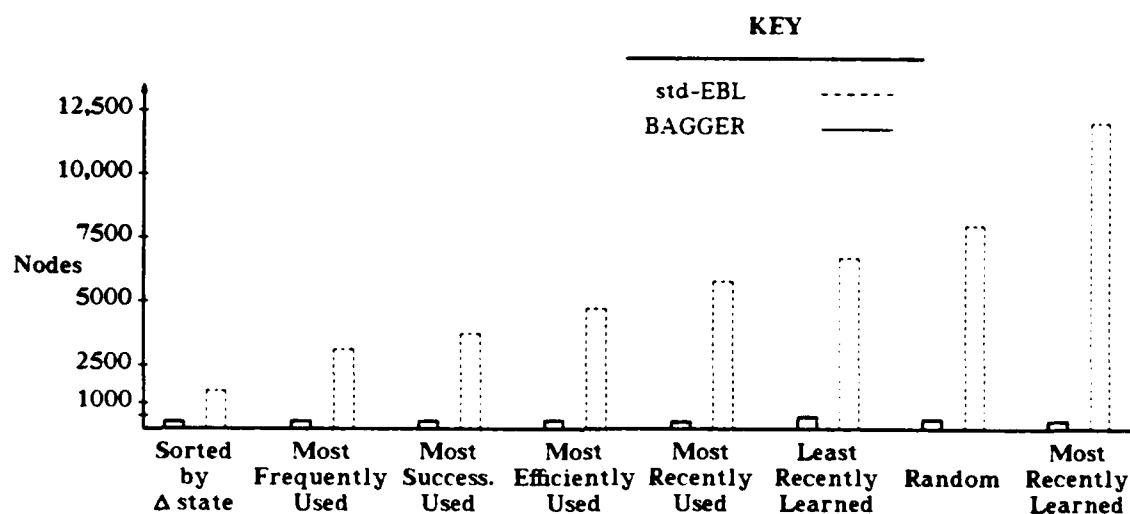


Figure 11.23 Performance of the Access Strategies

Many of the results in this experiment only differ by only a small amount. To eliminate the possibility that performances differed slightly due to machine differences, the timing data in this section is determined by counting the search tree nodes visited during problem solving. For *no-learn* this is estimated by extrapolating results on small problems in the same manner as done to estimate problem-solving time. The count of nodes is incremented whenever the consequent of a rule or an axiom unifies with the current goal.

To give a better perspective on the results in figure 11.23, in the next histogram (figure 11.24), the performances of the strategies are compared with respect to the performance of *MFU*. (The strategy involving sorting rules by the number of situations traversed (*SORT*) is not included in this figure because it is not relevant for **BAGGER**.) Relative performance is determined by:

$$\text{Relative Performance}(\text{strategy}) = \frac{\text{nodes}_{MFU}}{\text{nodes}_{\text{strategy}}}$$

Of the access strategies, the best performer for **EBL** is *SORT*. This occurs because the time to check the preconditions of a rule can increase exponentially with the number of situations traversed. *SORT* also works well because the most likely goal specifies the height of one average block.

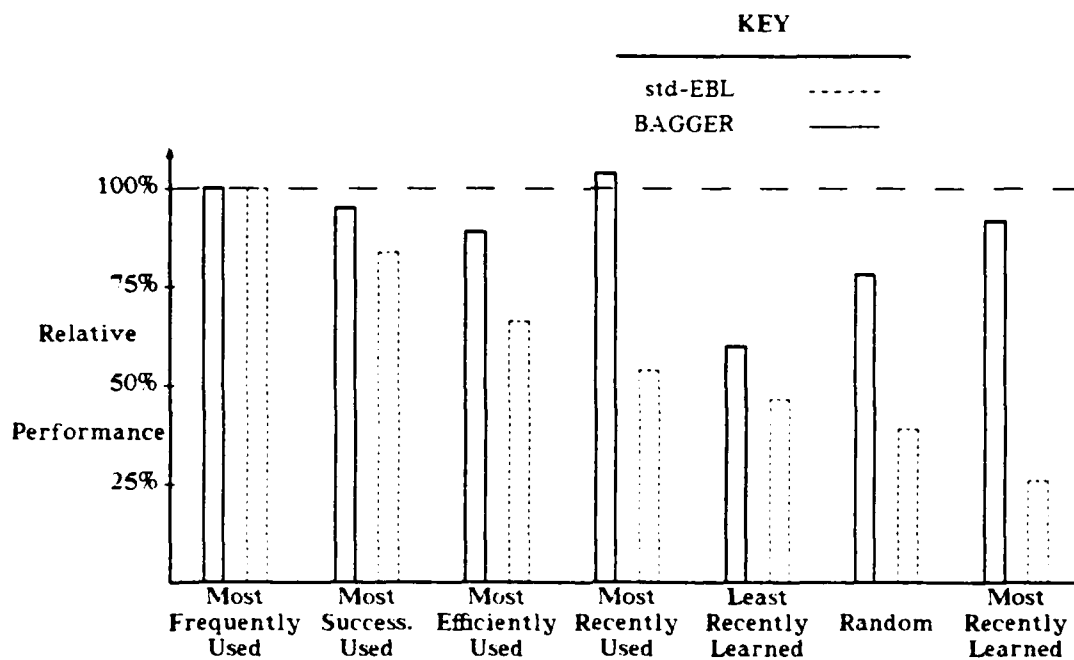


Figure 11.24 Performance of the Access Strategies Normalized Relative to MFU

The strategies that depend on previous problems, dynamically altering the order rules are accessed, prove beneficial. Keeping a count of the number of times a rule is successfully used works well, while the method of trying the most recently used rule first works less well. If the occurrence of a intricate, rare example requires the use of a complicated rule, moving it to the front of the queue may be a bad idea. On the next problem, substantial time may be spent discovering it is not applicable. It is better to have rules work their way forward by successfully solving several problems. However, as evidenced by *SORT*, it can be advantageous to first try rules whose preconditions can be rapidly checked, even if they are likely to solve fewer problems [Shavlik87e]. This topic is further discussed in section 11.9, where data on the distribution of solution lengths (in terms of the number of states traversed) is presented.

Somewhat surprisingly, while *LRL* works better than *MRL* for *sEBL*, the opposite is true for *BAGGER*. The reason for this is that earlier rules result from more typical examples, while later rules result from examples that are less likely to occur again soon. In *sEBL* it is best to first try the rules that result from the most probable situations. However, *BAGGER* often learns more from the more complicated, although less likely, examples and the acquired rule usually covers the simpler, more likely, examples. This indicates that, even if one of the dynamic

strategies is used, new rules should be added to the *end* of **sEBL**'s list of acquired rules, but they should be added to the *front* of **BAGGER**'s collection.

The next two figures present the relative speeds of the problem solvers under each strategy. Figure 11.25 reports the percentage of time each of the three problem solver is the fastest. In all cases, **BAGGER** solves more than half of the problems faster than do the other two systems. Except for *SORT*, **sEBL** solves 20-25% of the problems fastest. The reason for the near doubling of performance on *SORT* is that in this case **sEBL** never unsuccessfully tries to satisfy, for example, a rule that specifies the moving of four blocks, before trying a rule that moves only one. This gives it a better chance of outperforming **no-learn**.

The probability that **BAGGER** solves a problem faster than does **sEBL** is indicated by the data in figure 11.26. Except for *SORT*, **BAGGER** beats **sEBL** on about 70-75% of the test problems.

Each test set contains 20 problems. An interesting question is which access strategies organize rules so that performance improves as more and more problems are solved. The next histogram contains data relevant to this issue. Each strategy's performance on the first ten test problems is compared to its performance on the second ten. Figure 11.27 presents the results.

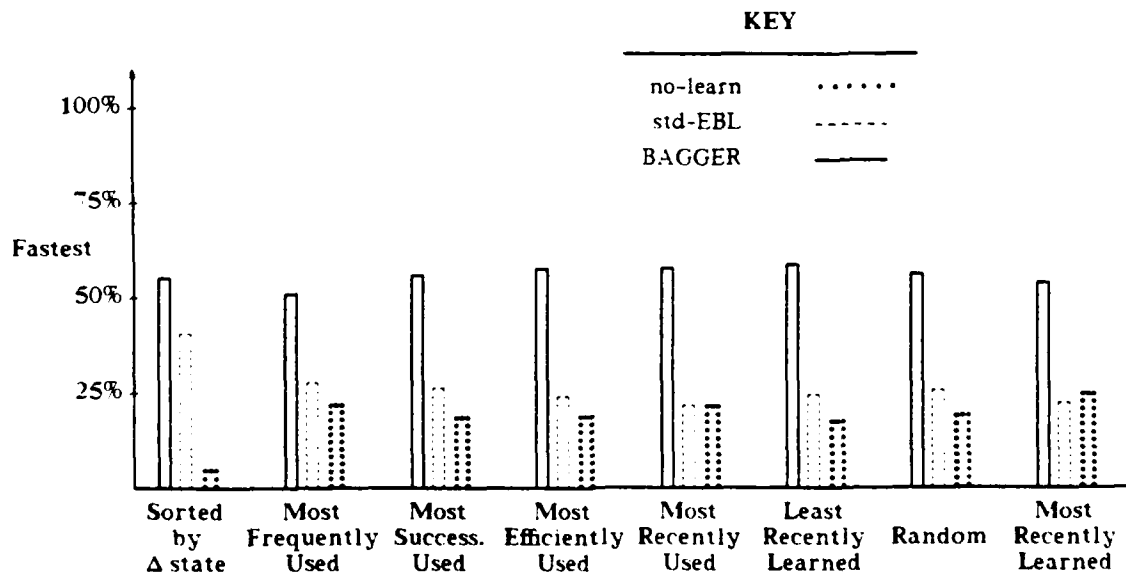


Figure 11.25 Relative Speeds for each Access Strategy

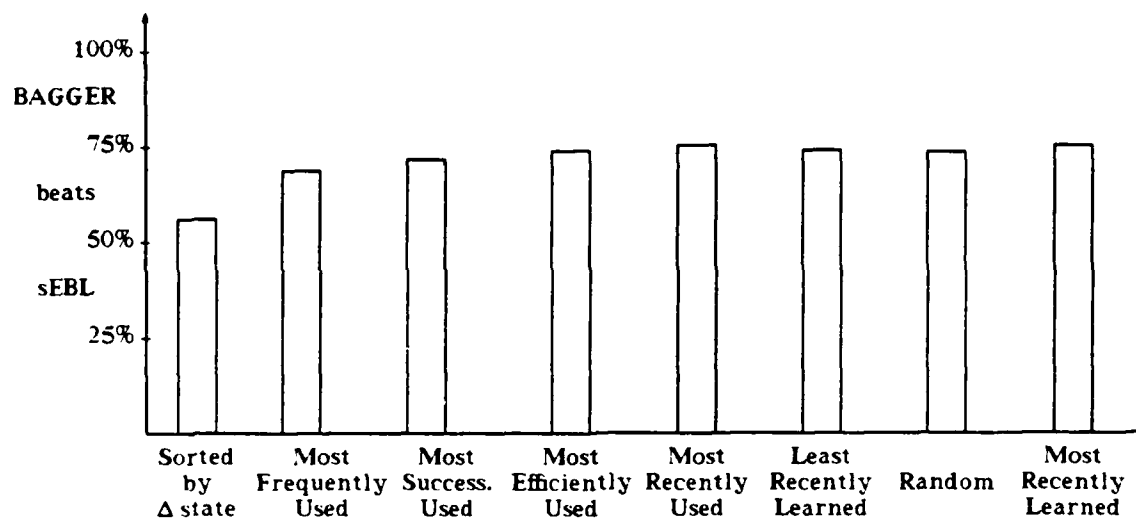


Figure 11.26 BAGGER Beats sEBL Percentage for each Access Strategy

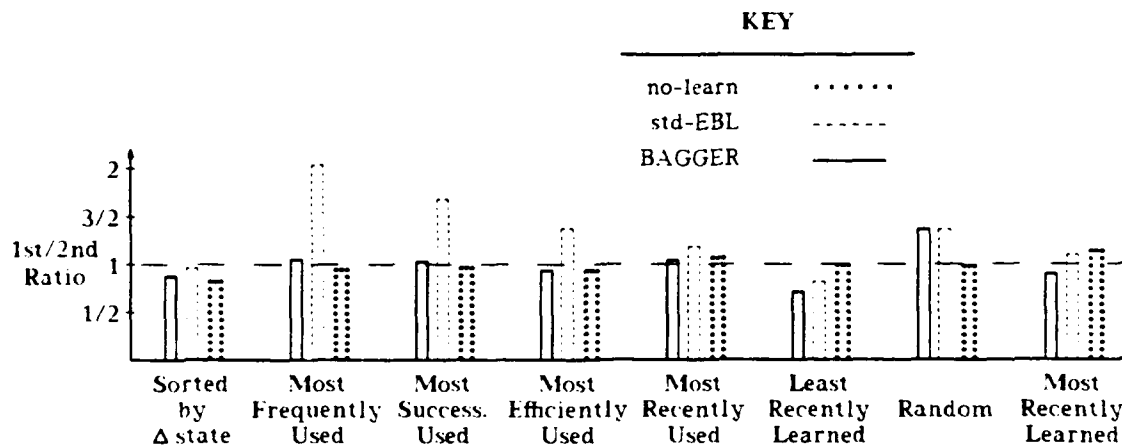


Figure 11.27 Temporal Improvement under each Access Strategy

The ratio of performance on the two groups of test problems is theoretically unity for **no-learn**, since it only depends on the random draw of desired tower heights. The ratio should also be unity for *SORT*, *LRL*, *RAND*, and *MRL*, because the order they try to apply rules is not effected by the results of previous test problems. The heights of these bars provide an indication of the variability due to statistical fluctuations. This figure demonstrates that the strategies in which rules are continually reorganized during problem solving improve performance, especially in sEBL, where more rules are learned on average.

Figure 11.28 shows for each access strategy the probability, given a rule is tried, that it will prove successful. This probability is defined in the previous section. Again, the cases where rules are reorganized during the test phase perform somewhat better.

The final question investigated in this section is another measure of the efficiency of the access strategies. Here efficiency is measured in terms of the number of actions required to execute the plans produced. If the functions that cause situation changes represent actions that have to be executed in the external world, it may be important to minimize the number of situations traversed by a rule application. Figure 11.29 shows the mean number of blocks moved under the various conditions. By its definition, *SORT* (in combination with *sEBL*) will provide the shortest solution possible, given a collection of rules. In all of the other cases, *BAGGER* outperforms *sEBL*.

As stated earlier, the performance of *no-learn* is estimated by extrapolation and its solutions are produced by a specialized procedure. This procedure selects blocks to add to the tower in reverse order of the way they are dropped (e.g., block 10 is chosen first, then block 9, etc.). Blocks are chosen until the sum of their heights satisfies the goal. This method insures

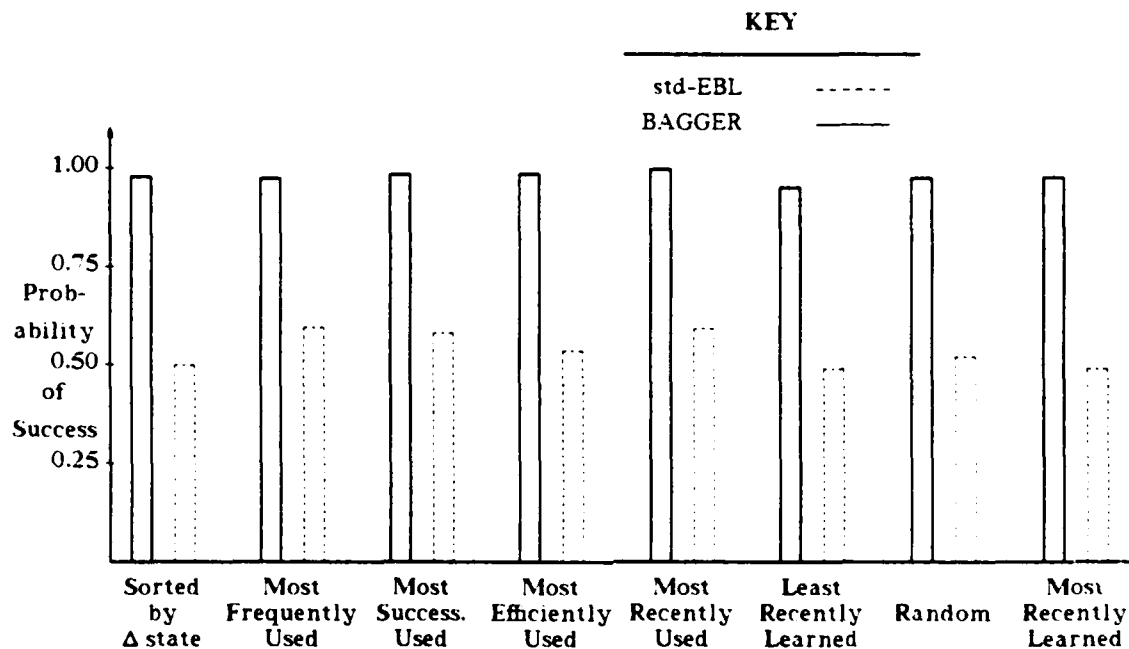


Figure 11.28 Probability of Successful Rule Application under Each Access Strategy

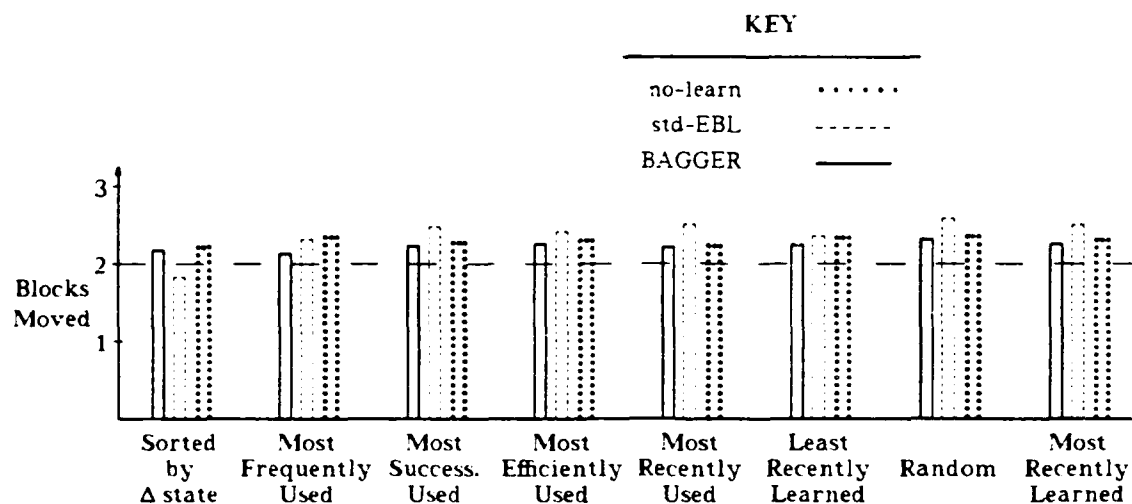


Figure 11.29 Mean Number of Blocks Moved under each Access Strategy

that a block is always clear when it is chosen to be moved. However, it does not produce the shortest possible solution.

The main reason **sEBL** is outperformed is that a rule for moving three blocks, for instance, can satisfy a wide range of tower heights, since the height of individual blocks is randomly determined. Hence, often three small blocks will be moved when one or two larger ones would suffice. More information on the topic of solution lengths is reported in section 11.9.

The experiments in this section investigate various strategies for determining the order to access rules. The results also provide additional evidence of the value of **BAGGER** over standard explanation-based learning. Under various strategies for deploying rules, **BAGGER** always outperforms **sEBL**. The most successful access strategy for **sEBL** is to sort rules according to the number of states traversed, a technique applicable if situation calculus is being used. If this type of sorting is not possible, the order of rule access should depend on the examples experienced. Trying the most frequently successful rule first substantially improves performance. Finally, for **sEBL** new rules should be added to the *end* of the list of acquired rules, while for **BAGGER** they should be added to the *front*.

11.7. Combining sEBL and BAGGER

When **BAGGER** cannot produce a sequential rule from an explanation, nothing is learned. This section investigates the effect of combining the two learning systems, that is, applying **sEBL** whenever **BAGGER** fails to produce a new rule. In this case, rules from both systems are stored in the same list.

Figures 11.30 and 11.31 compare this extended version of **BAGGER** in the autonomous and training modes, respectively. In the autonomous mode, there is substantial improvement. This is due to the fact that the rules produced by **sEBL** decrease the likelihood that **BAGGER** will have to resort to first principles to solve a problem. Since a large amount of time is spent building the explanation from which the **BAGGER** algorithm could not produce any new rule, it is worthwhile to apply **sEBL** to the constructed explanation, rather than merely discarding it. If the same or a similar problem occurs later, a rule for solving it will be possessed.

However, in training mode incorporating **sEBL** degrades **BAGGER**'s performance. With a large enough training set, it is likely that if one sample solution is not sufficient to learn from, a later solution will lead to a rule that also solves the first problem. For example, **BAGGER** does

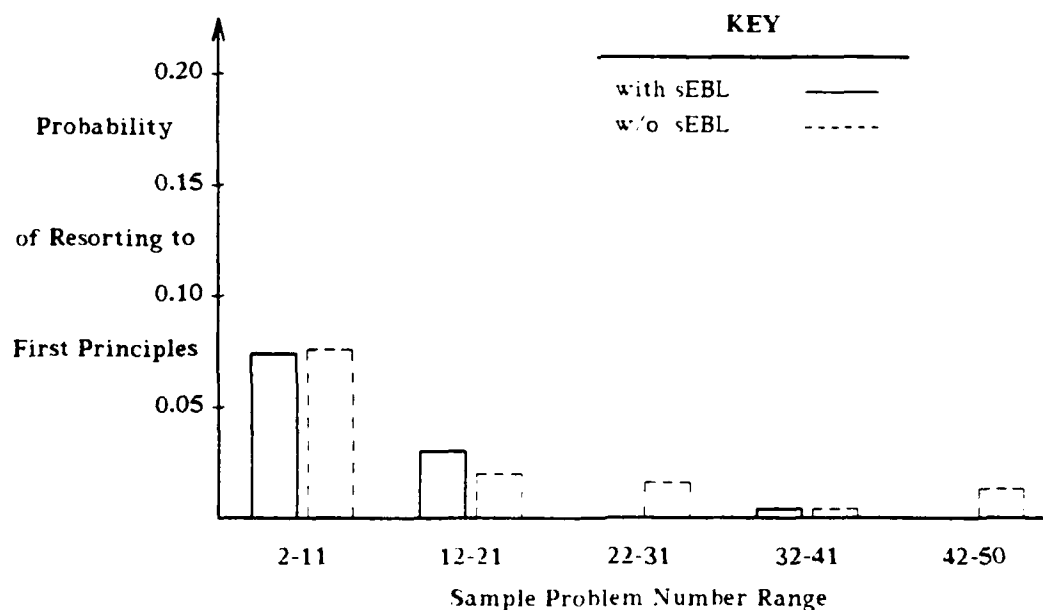


Figure 11.30 Combining sEBL and BAGGER in Autonomous Mode

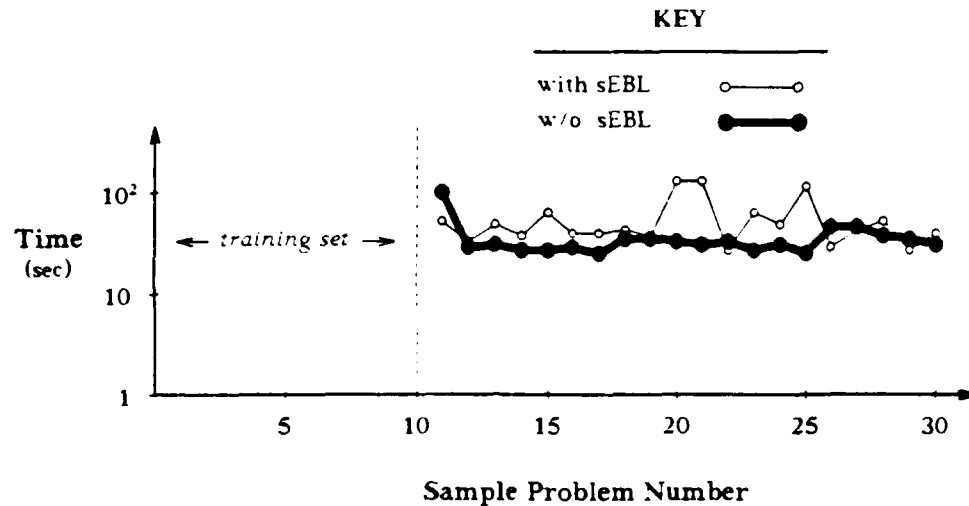


Figure 11.31 Combining sEBL and BAGGER in Training Mode

not learn from the construction of a one-block tower. But if it later experiences the construction of a three-block tower, it learns a rule that can be used to construct towers of any height. Since in the training mode explanation construction is relatively inexpensive, it is less harmful to discard one in the hopes of a better one coming along later.

The above argument requires that the training set be adequate to insure that rules learned by **BAGGER** cover most of the problems to be experienced in the future. If the training set is too small or if the solutions to many possible problems cannot be generalized by **BAGGER**, it would be advantageous to combine sEBL and **BAGGER**. Basically, it is a question involving the trading of longer solution times for higher probabilities of successful solutions.

The next two figures show the how the number of rules learned by **BAGGER** is affected by incorporating sEBL. Figure 11.32 shows there is only a slight increase in the autonomous mode. Conversely, figure 11.33 shows that in the training mode there is a substantial increase. This is due to the fact that in the training mode previously acquired rules are not tested to see if they solve the current training problem before generalization occurs. Hence, even if a previous **BAGGER** rule covers the current training problem, a new sEBL rule may be learned.

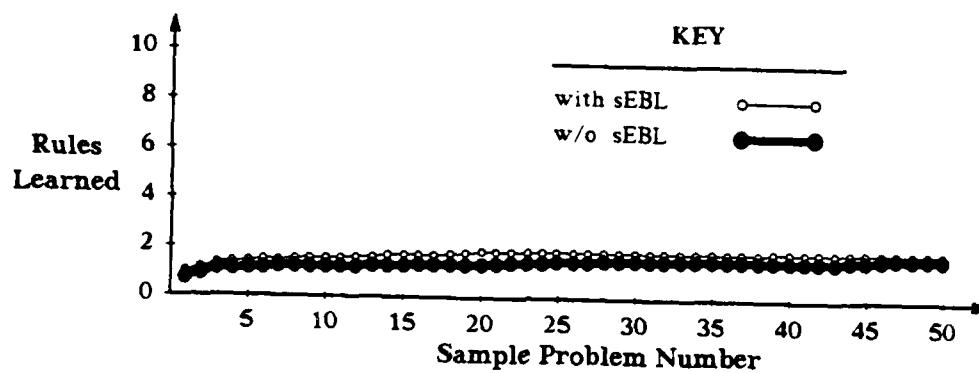


Figure 11.32 Rules Learned when Combining sEBL and BAGGER in Autonomous Mode

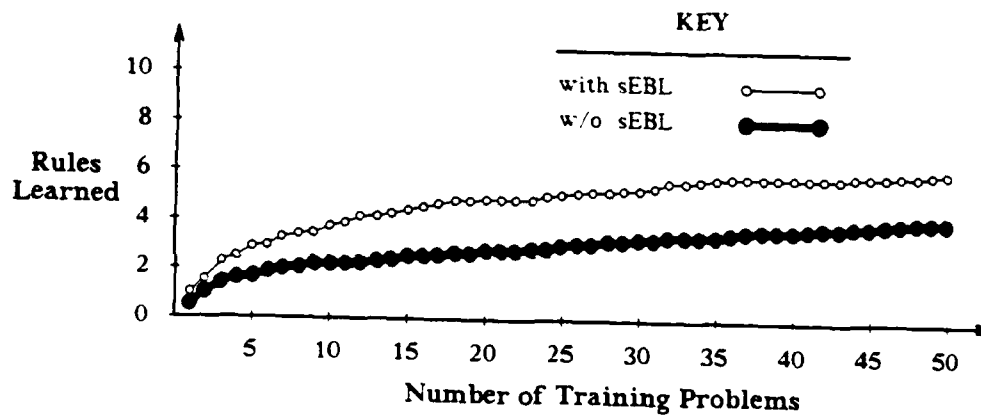


Figure 11.33 Rules Learned when Combining sEBL and BAGGER in Training Mode

Tables 11.7 and 11.8 present the relative speed summaries for these experiments. Here BAGGER incorporates sEBL when necessary, while the usual stand-alone sEBL system also operates, for purposes of comparison. The measurements are comparable in the two cases.

The results of this experiment are summarized in table 11.9. They indicate that combining BAGGER and sEBL is appropriate in the autonomous mode and is inappropriate in the training mode.

Table 11.7 Relative Speed Summary in Autonomous Mode
with BAGGER and sEBL Combined

	1st	2nd	3rd
<i>No-Learn</i>	19.2% (20.2) ⁶	19.0 (22.9)	61.8 (57.0)
<i>Std-EBL</i>	27.5 (24.8)	40.7 (41.6)	31.8 (33.6)
<i>BAGGER</i>	53.3 (55.0)	40.3 (35.5)	6.4 (9.4)

BAGGER beats standard EBL: 70.6% (71.8)

Table 11.8 Relative Speed Summary in Training Mode
with BAGGER and sEBL Combined

	1st	2nd	3rd
<i>No-Learn</i>	17.6% (20.8)	15.0 (15.1)	67.4 (64.1)
<i>Std-EBL</i>	26.9 (21.6)	44.1 (47.8)	29.0 (30.7)
<i>BAGGER</i>	55.5 (57.6)	40.9 (37.1)	3.6 (5.2)

BAGGER beats standard EBL: 71.2% (75.4)

⁶ In this table and the one below, the number in parentheses refer to the results from the previous experiments where **BAGGER** does not use sEBL.

Table 11.9 Value of Combining BAGGER and sEBL	
Description	Mean Value
Solution Time	
<i>Autonomous</i>	
w/o std-EBL	3720.0 sec
with std-EBL	57.5
<i>Trained</i>	
w/o std-EBL	36.6
with std-EBL	46.2
Rules Learned	
<i>Autonomous</i>	
w/o std-EBL	1.72
with std-EBL	1.96(0.37) ⁷
<i>Trained</i>	
w/o std-EBL	2.00
with std-EBL	3.89(1.56)
Faster than sEBL	
<i>Autonomous</i>	
w/o std-EBL	71.8%
with std-EBL	70.6
<i>Trained</i>	
w/o std-EBL	75.4
with std-EBL	71.2

11.8. Clearing Blocks

This section reports the performance of the three systems on a different problem. In this case, initial situations are constructed in the same manner as for tower-building, but the goal instead specifies a block to be cleared. After the ten blocks are randomly dropped, one of the unclear blocks is randomly chosen as the goal block to be cleared. To keep the two experiments comparable, if the goal block supports, directly or indirectly, more than four other blocks a new goal block is chosen.

⁷ Results in parentheses indicate the mean number of standard EBL rules learned. The other number is the total of BAGGER and standard EBL rules.

Figure 11.34 contains the performance, in the training mode, of the three systems on this task. In this case, the learning systems outperform **no-learn** by 4-5 orders of magnitude, several orders of magnitude better than in the tower-building experiments. One interpretation of this is that clearing is a much more "local" operation than is tower building. Clearing a block involves moving nearby blocks, blocks coupled to the goal block by relations such as *On* and *Supports*. However, when building a tower, blocks located anywhere in a scene can be used. In general there is less to constrain the choice of a block to move when building a tower.⁸ For these reasons, the advantages of explanation-based learning, which collects the essential constraints in an example, become more pronounced.

The relative speeds of the three systems for this problem appear in table 11.10. Corresponding to the results in figure 11.34, **no-learn** is usually outperformed by the two learning systems.

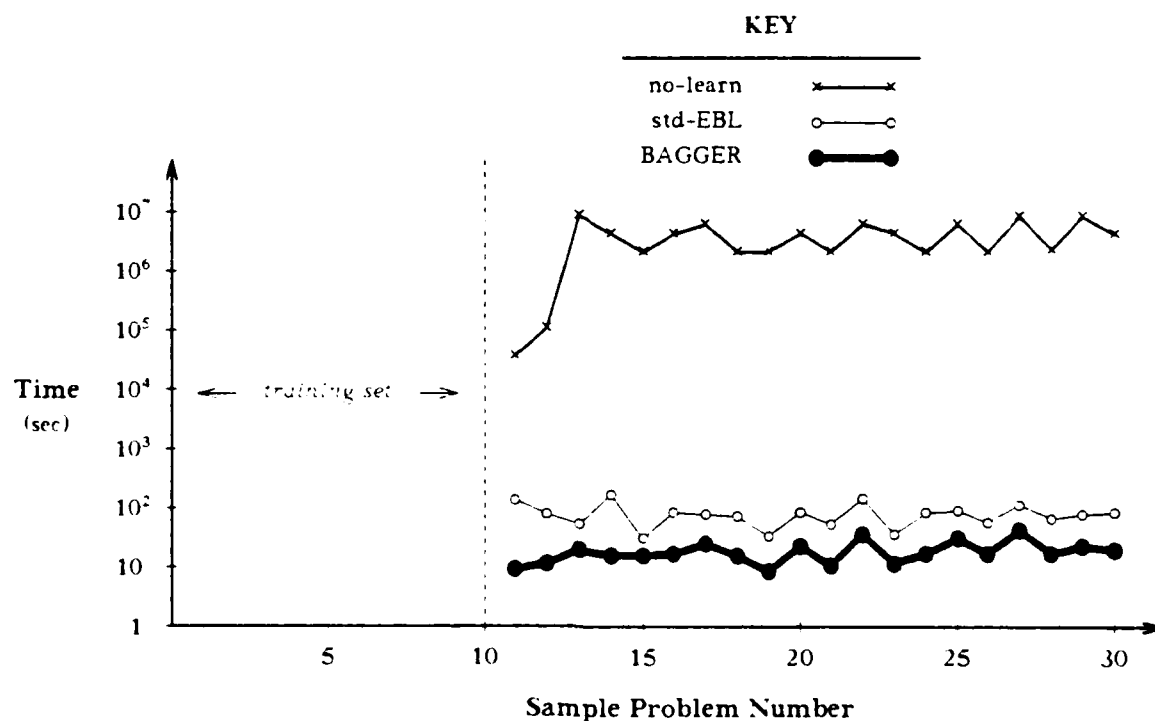


Figure 11.34 Performance Results for Clearing Blocks

⁸ This would be less true if the acquired rules were complicated enough to explicitly represent how much of the tower height remained after each move.

Table 11.10 Relative Speed Summary for Clearing

	1st	2nd	3rd
<i>No-Learn</i>	0.0% (20.2) ⁹	14.2 (22.9)	85.8 (57.0)
<i>Std-EBL</i>	34.7 (24.8)	51.1 (41.6)	14.2 (33.6)
<i>BAGGER</i>	65.3 (55.0)	34.7 (35.5)	0.0 (9.4)

BAGGER beats standard EBL: 65.3% (71.8)

Table 11.11 presents a comparison of the three problem solvers on the two types of problems. The speed-up of **BAGGER** over **sEBL** is about one-fifth for clearing as it is for building towers. This may be due to the fact that, on average, less blocks are moved in the clearing problems.¹⁰ As reported in section 11.3, the speed-up of **BAGGER** over **sEBL** increases as the complexity of the problem space increases.

Table 11.11 also contains the results from an experiment where *both* clearing and tower-building goals are generated. The type of goal is chosen randomly and the training set contains twenty problems, rather than the usual ten, so that on average ten of each type are experienced. The results are largely the average of those from the two types of experiments, except that more rules are learned and the probability of a successful rule application is decreased. The extra rules do not significantly hinder the problem solvers because the consequents of the two rule types are different enough that a rule for the wrong goal can be quickly disregarded.

The reason for the different solution lengths of the three problem solvers for clearing is that solutions lengths are only computed for problems successfully solved. The results indicate that the learners are more likely to fail on problems requiring more blocks to be moved and that **BAGGER** can solve, on average, more complicated problems than can **sEBL**.

These experiments involving clearing provide further support for the claims that explanation-based learning is beneficial and that **BAGGER** outperforms **sEBL**.

⁹ The numbers in parentheses are for tower-building.

¹⁰ The distributions of solution lengths for the various experiments are reported in the next section.

Table 11.11 Comparison of Tower Building and Clearing in Training Mode			
Description	No-Learn	Std-EBL	BAGGER
Mean Solution Time			
<i>Clearing</i>	4,200,000 sec	58(86) ¹¹	15(20)
<i>Tower-Building</i>	68,400	779(828)	36(37)
<i>Both</i>	1,790,000	601(607)	31(33)
Speed-Up Over No-Learn			
<i>Clearing</i>	—	72,400	280,000
<i>Tower-Building</i>	—	88	1,900
<i>Both</i>	—	2,980	57,700
Fastest			
<i>Clearing</i>	0.0%	34.7	65.3
<i>Tower-Building</i>	20.8	21.6	57.6
<i>Both</i>	9.2	31.7	59.1
Percentage Solved			
<i>Clearing</i>	100.0%	84.6	85.2
<i>Tower-Building</i>	100.0	98.5	99.4
<i>Both</i>	100.0	93.4	94.8
Solution Length			
<i>Clearing</i>	1.87	1.65	1.77
<i>Tower-Building</i>	2.20	2.51	2.22
<i>Both</i>	1.97	2.03	1.97
Rules Learned			
<i>Clearing</i>	0.00	3.96	1.36
<i>Tower-Building</i>	0.00	5.96	2.00
<i>Both</i>	0.00	9.09	3.59
Probability of Successful Application			
<i>Clearing</i>	—	0.37	0.81
<i>Tower-Building</i>	—	0.59	0.99
<i>Both</i>	—	0.25	0.60

¹¹ Numbers in parentheses indicate mean solution time for all problems, including problems not solved.

11.9. Additional Experimental Results

This section presents and analyzes additional experimental details. First, several alternative ways to train a system are considered. Next, the cost of excessive training is investigated. Following that the mean number of rules applied to a problem is reported for the two modes of operation. After that, the value of heuristically reorganizing antecedents during learning is investigated and the probability that **BAGGER** learns a rule that does not solve the problem form which it is learned is estimated. Finally, before reporting the estimates of the performance of **no-learn**, the distribution of solution lengths under various conditions is discussed.

Alternative Modes of Operation

The two modes of operation investigated — autonomous and training — are at opposite ends of a spectrum. In one, the learning systems are completely independent of external guidance. In this mode, learning occurs whenever none of the acquired rules suffice to solve the current problem, while in the other all of the learning occurs during an initial training phase, where sample solutions are provided by an external agent. The cost of producing explanations from first principles in the autonomous mode can be prohibitive. Conversely, the training set of problems may not be fully representative of future problems, which means that some future problems may not be soluble.

This section considers some ways of combining the strengths of the two approaches. The results are especially relevant to the design of expert systems that dynamically acquire new rules by observing expert behavior. Consider the following two ways to utilize external expertise that are alternatives to the training mode.

- (1) The expert can be continually on call. When one of the learning systems cannot solve a problem using its collection of acquired rules, the expert produces a solution. This solution is then explained and any new rule that results is added to the collection of acquired rules. As more rules are learned, the mean time between requests for the expert will increase.
- (2) Again an expert is always available, except here he is not called until a time threshold is exceeded by one of the three systems. Time can be spent both checking acquired rules and building solutions from first principles. With this method, relatively simple problems can

be solved from first principles, thereby decreasing the load on the expert, especially during the early stages of learning. However if the threshold is set too low, the expert will be unnecessarily called on problems that are soluble by the acquired rules.

These techniques mitigate the expense of solving from first principles. If the failure to solve a problem is unacceptable, combinations of the above techniques with the training mode are also possible. For example, the expert can provide solutions to an initial set of problems, then provide additional solutions during the test phase whenever the rules learned during the training phase prove insufficient. Alternatively, during the test phase solutions can be derived from first principles as done in the autonomous mode.

Table 11.12 compares the results of these approaches with the standard autonomous and training modes. All of the averages under the autonomous mode are for problems 26–50. **BAGGER**'s performance, with and without combining it with **sEBL**, is also reported. An interesting point is that the two hybrid approaches perform about equally well for **sEBL**. This occurs because it is possible for **sEBL** to spend more than 10,000 seconds checking its acquired rules.

The results in this section indicate that these hybrid approaches significantly exceed the performance of the fully autonomous approach, while still guaranteeing solutions to all problems. The cost of this is that an expert must be available at any time, although as time progresses he should be needed less frequently.

Cost of Excessive Training

In the training mode, increasing the size of the training set decreases the probability that a test problem will not be solved. However there are three costs of increasing the training set size. One, the training period takes longer to complete, two, more rules will be acquired, and, three, more time can be spent applying the acquired rules. The law of diminishing returns applies here. Incremental gains in the probability of future success may not be worth the increase in the mean time to solve a problem.

Figure 11.35 presents data relevant to this issue. A measure of average performance on 20 test problems is plotted as a function of the size of the training set. Results are presented for **sEBL** under two conditions. As they are learned, new rules are either added to the front or to

Table 11.12 Comparison of Operation Modes	
Mode of Operation	Mean Solution Time (sec)
Autonomous <i>Std-EBL</i> BAGGER w/o std-EBL with std-EBL	8100(7580) ¹² 3720 58
Autonomous <i>(external solutions after 10,000 sec)</i> <i>Std-EBL</i> BAGGER w/o std-EBL with std-EBL	1290(949) 133 52
Autonomous <i>(external solutions if all rules fail)</i> <i>Std-EBL</i> BAGGER w/o std-EBL with std-EBL	1470(920) 40 36
Trained <i>Std-EBL</i> BAGGER w/o std-EBL with std-EBL	827(988) 37 46

¹² Figures in parentheses are from the runs where **BAGGER** used standard EBL. There is no theoretical reason the two numbers should differ other than due to statistical fluctuations.

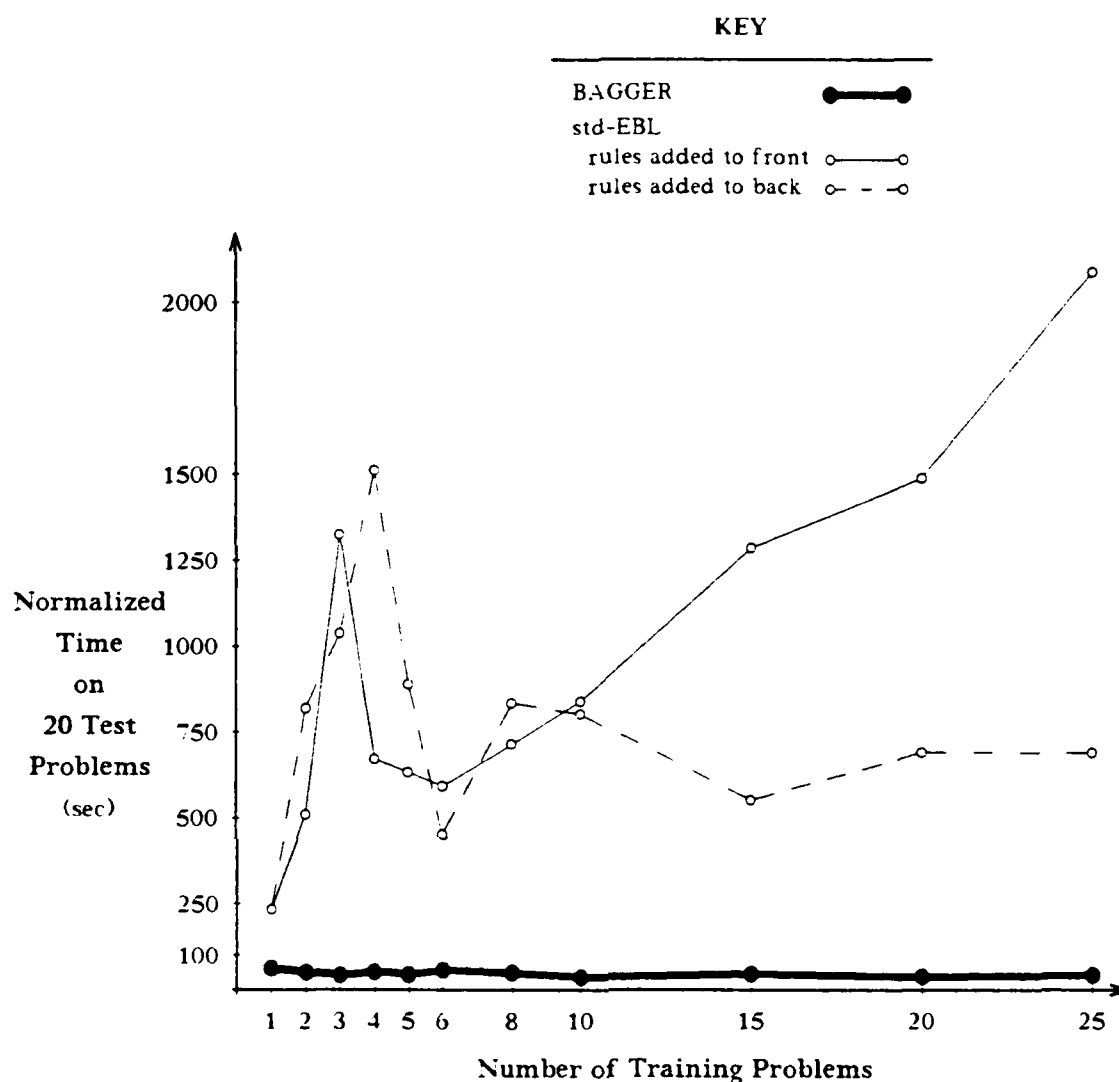


Figure 11.35 Effect of Excessive Training

the back of the list of previously-acquired rules. Section 11.6 discussed the value to *sEBL* of considering rules in the order they are learned. In all cases, the *MRU* strategy is used during the test phase. Normalized solution time, defined below, is plotted as a function of the training set size.

$$\text{normalized solution time} = \frac{\text{mean solution time}}{\text{probability of successful solution}}$$

Normalizing by the probability of a successful solution partially compensates for the fact that quickly solving only a few problems is not desirable.

This figure shows that too much training can be detrimental. Due to the broad coverage of the rules it learns, **BAGGER** is not affected by the additional training. But excessive training clearly causes a decrease in the performance of **sEBL** if the most recently learned rules are tried first. If the training mode is used, care must be taken in choosing the size of the training set.

Rules Tried

In this section, the mean number of rules tried when addressing a problem is plotted for the two modes of operation. Figure 11.36 shows that in the autonomous mode **sEBL** tries slightly more rules as problem solving progresses, while **BAGGER** quickly acquires a rule that suffices for the rest of the problems. The rules tried in the training mode are plotted in figure 11.37. Again **sEBL** tries about $1\frac{1}{2}$ rules in order to solve each problem.

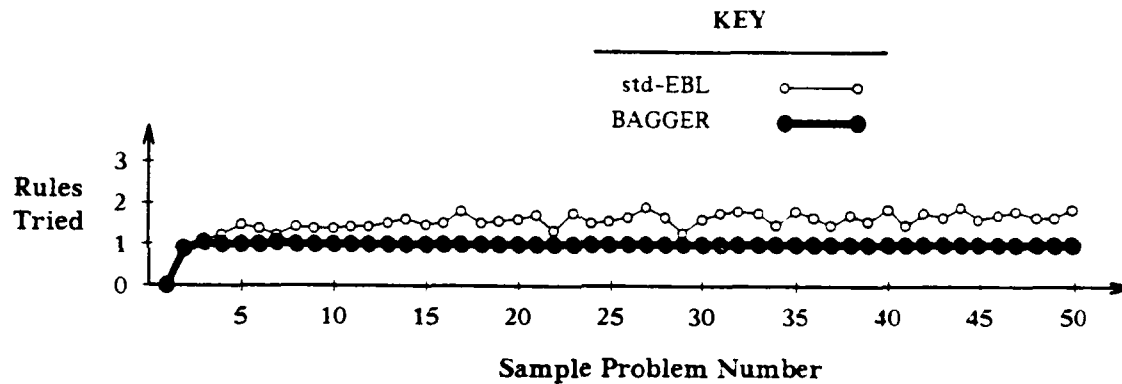


Figure 11.36 Rules Tried in Autonomous Mode

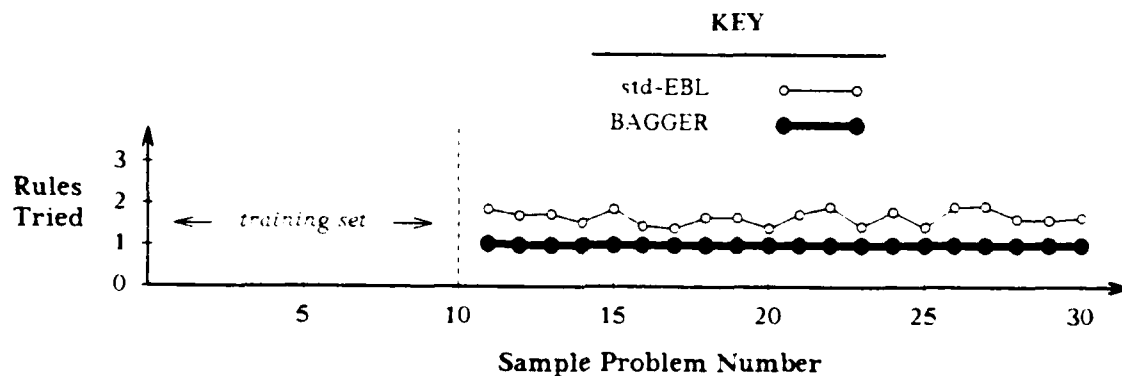


Figure 11.37 Rules Tried in Training Mode

Relearning the Same Rule in BAGGER

As discussed in chapter 9, it is possible that **BAGGER** can learn a rule that does not apply to the problem from which the rule is learned. The frequency that this occurs is estimated by determining how often a rule learned in the autonomous mode is a variant of a rule already possessed. Since the variant rule failed to solve the current problem, so will the new one. Learning a variant occurred in about 5% of the cases where **BAGGER** could not solve the current problem in the autonomous mode.

Solution Lengths

As discussed earlier, minimizing the number of actions in a solution may be of importance. This section reports statistics on solution lengths, in terms of the number of situations traversed, for various experiments. This data also sheds further light on the behavior exhibited in the experiments. Tables 11.13-11.16 report on experiments involving the building of towers, while table 11.16 refers to the experiment involving the clearing of blocks. These tables report, for the various numbers of blocks moved, the mean solution time (in seconds) and the number of times this many blocks are moved (this is the number in parentheses). In the autonomous mode the data under "0" reflects the cases where the acquired rules failed to solve the current problem (in problems 26-50). The reported time in this column includes the time spent solving from first principles. In the tables for the training mode, the first column reflects the number of times no solution is found and the time spent exhausting all of the acquired rules when this occurs. Note that the statistics in column *N* do not directly reflect how much time is spent only on **SEBL** rules that traverse *N* states. Rather, they include how much time is spent trying *all* rules before finding a solution of length *N*.

There are several observations about these tables. First, the **SEBL** entry in the table for the *SORT* access strategy gives an indication of the optimal solution lengths. (**BAGGER** is unaffected by this strategy. That is, the data for **BAGGER** should be identical in the table for *SORT* and the table above it. Differences are due to statistical fluctuations.) The table for *SORT* indicates that, although **BAGGER** does not produce optimal solutions, it comes close.

Second, there are a few points about the table for clearing. The reason that on fifteen occasions **BAGGER** could not clear a block supporting only one other block is that on one experimental run **BAGGER** learned no rules from the training set, which accounts for 4% of its

Table 11.13 Time/Occurrences v. Blocks Moved in Autonomous Mode

	0	1	2	3	4	5
<i>No-Learn</i>	0(0)	11(222)	383(156)	13,229(143)	457,854(104)	0(0)
<i>Std-EBL</i>	251,058(17)	1514(179)	1391(148)	870(128)	1363(153)	0(0)
<i>BAGGER</i> ¹³	384,065(6)	22(215)	24(162)	42(132)	70(92)	156(17)

Table 11.14 Time/Occurrences v. Blocks Moved in Training Mode

	0	1	2	3	4	5
<i>No-Learn</i>	0(0)	11(166)	382(140)	13,229(123)	457,854(71)	0(0)
<i>Std-EBL</i>	12,810(2)	1507(65)	879(211)	431(126)	524(96)	0(0)
<i>BAGGER</i> ¹⁴	251(2)	21(168)	26(146)	34(103)	57(72)	311(8)

Table 11.15 Time/Occurrences v. Blocks Moved for *SORT* in Training Mode

	0	1	2	3	4	5
<i>No-Learn</i>	0(0)	11(181)	382(131)	13,229(102)	457,854(86)	0(0)
<i>Std-EBL</i>	2110(13)	8(217)	89(141)	448(101)	1197(28)	0(0)
<i>BAGGER</i>	1254(5)	20(178)	27(140)	41(97)	77(65)	138(15)

Table 11.16 Time/Occurrences v. Blocks Moved to Clear in Training Mode

	0	1	2	3	4	5
<i>No-Learn</i>	0(0)	31(226)	3692(148)	447,232(83)	5×10^7 (43)	0(0)
<i>Std-EBL</i>	321(77)	46(226)	80(127)	136(64)	164(6)	0(0)
<i>BAGGER</i>	109(74)	8(209)	17(125)	61(64)	204(28)	0(0)

¹³ On one example *BAGGER* took 209 seconds and moved six blocks.

¹⁴ On one example *BAGGER* took 77 seconds and moved six blocks.

failure rate of 14.8%. Other failures are largely due to the fact, described in the previous chapter, that **BAGGER** can only learn what are termed horizontal and vertical clearing.

Finally, the tables indicate that in **sEBL** rules for moving three blocks are most likely to be tried first when building towers. This occurs because, due to the variance in block heights, a rule for moving three blocks can satisfy goal heights constructed from two, three, or four average blocks, possibly even one. Recall that the average goal height is equal to the height of two average blocks. This illustrates the possibility that a rule which takes substantial time to satisfy may move to the head of the rule queue because it solves a large collection of problem, thereby degrading overall performance.

Estimating the Performance of the No-Learn System

Due to computational resource limits, the performance of **no-learn** is estimated. The equations used to estimate the performance of **no-learn** appear in figure 11.38. N is the number of blocks to be moved. These equations are constructed by measuring the system's performance on 250 problems where N equals one or two, then using the curve fitting algorithm of section 14.2 in [Press86]. Problems where N equals three occasionally exhaust the available memory. The successful cases with $N=3$ are consistent with the estimations.

$$time_{tower}(N) = 0.32 \times 10^{1.54 \times N} \text{ seconds}$$

$$time_{clear}(N) = 0.25 \times 10^{2.69 \times N} \text{ seconds}$$

$$nodes_{tower}(N) = 5.45 \times 10^{1.35 \times N}$$

$$nodes_{clear}(N) = 2.68 \times 10^{1.98 \times N}$$

Figure 11.38 Estimated Performance of No-Learn

11.10. Summary

The empirical results presented in this chapter demonstrate the value of generalizing the structure of explanations in particular, and explanation-based learning in general. Three different problem solvers are compared. Two of them learn: **BAGGER** and **sEBL** (an implementation of the **EGGS** [Mooney86] explanation-based generalization algorithm). The other, called **no-learn**, performs no learning. In the experiments investigated, **BAGGER** and **sEBL** perform substantially better than the system that performs no learning, by up to five orders of magnitude. **BAGGER** also outperforms the standard EBL system.

Investigation of two training modes demonstrates the importance of external guidance to learning systems. In the autonomous mode, the systems must solve all problems on their own. The high cost of doing this when no learned rule applies dissipates much of the gains from learning. Substantial gains can be achieved by initially providing solutions to a collection of sample problems, and having the learners acquire their rules by generalizing these solutions. The usefulness of this depends on how representative the training samples are of future problems and how acceptable are occasional failures. Since **BAGGER** requires less training examples and produces more general rules, it addresses these issues better than does standard explanation-based learning.

The **BAGGER** algorithm leads to the acquisition of fewer rules, because one of its rules may subsume many related rules learned using standard explanation-based learning. In this chapter, experiments demonstrate that acquiring fewer rules decreases the likelihood that time will be wasted on rules that appear to be applicable. The probability that, in the training mode, a retrieved rule successfully solves a tower-building problem is about 0.60 for **sEBL** and 0.99 for **BAGGER**.

A number of additional statistics are reported. The time **BAGGER** takes to generalize an explanation is comparable to that for **sEBL**. Eight strategies for determining the order to access acquired rules are analyzed. Dynamically organizing rules so the most recently or frequently used ones are tried first proves beneficial. If situation calculus is used, **sEBL** benefits most by ordering rules according to the number of situations they traverse. For **BAGGER**, new rules should be added to the front of its collection of rules, while for **sEBL** they should be added to the back. Rules that are more operational perform better when the training mode is used.

Combining **BAGGER** and **sEBL** can be beneficial, especially in the autonomous mode. Additionally, fewer training examples are needed for **BAGGER** to acquire a sufficient set of new rules and excessive training examples impact **BAGGER** less severely than **sEBL**. The advantages of **BAGGER** over standard explanation-based learning magnify as the range of potential problems is increased.

Other researchers have also reported on the performance improvement of standard explanation-based learning systems over problem solvers that do not learn [Fikes72, Minton85, Mooney88, O'Rourke87b, Prieditis87, Steier87]. One major issue is that, as more new concepts are learned, problem-solving performance can *decrease*. This occurs because substantial time can be spent trying to apply rules that appear promising, but ultimately fail [Minton85, Mooney88]. While the non-learning system outperforms the learning systems on some problems, in the experiments reported in this chapter the overall effect is that learning is beneficial. Significantly, as the complexity of the problem space increases, so does the improvement achieved by explanation-based learning.

Another potential performance degradation can occur when a new broadly-applicable rule, which can require substantial time to instantiate, blocks access to a more restricted, yet often sufficient, rule whose preconditions are easier to evaluate [Shavlik87e]. (Also see chapter 4.4.) Evidence for this is found in the experiments involving the **sEBL** system. When building towers that contain on average about two blocks, rules that specify moving three blocks are preferentially chosen. This occurs because these three-block rules, while less efficient than rules for moving two blocks, cover a larger collection of the possible problems.

While this chapter's experiments indicate that learning increases overall problem-solving performance, evidence is also presented (section 11.9) that too much learning can be detrimental. Techniques for forgetting or reorganizing acquired rules are necessary. The experiments investigating rule access strategies (section 11.6) demonstrate the value of dynamically organizing rules. The idea of organizing acquired rules so that rapidly evaluated special cases of more general rules are tried first is presented in section 4.4 and [Shavlik87e]. An untested approach to forgetting is presented in [Fikes72], where it is suggested that statistics be kept on the frequencies acquired rules are learned, discarding those that fall below some threshold. This idea is successfully tested in [Minton85]. To prevent the accumulation of excessive number of rules, it is also important to wisely decide when to acquire a new rule. Some general heuristics

have been proposed that estimate when it is a good idea to construct the generalization of a specific explanation. For example, [Minton85] proposes selectively generalizing those solutions that initially progress away from the direction indicated by a hill-climbing measure. A similar idea appears in [Iba85]. Only solutions that achieve *thematic* goals [Schank77] are generalized in the GENESIS system [Mooney85]. Additional conditions for generalization are presented in [DeJong83].

It may seem that largely investigating only tower-building problems unfairly favors explanation-based learning. An alternative would be to investigate a more diverse collection of problem types. However, the negative effects of learning are manifested most strongly when the acquired concepts are closely related. If the effects of some rule support the satisfaction of a goal, substantial time can be spent trying to satisfy the preconditions of the rule. If this cannot be done, much time is wasted. To the sEBL system, a rule for stacking two blocks is quite different from one that moves four blocks. Frequently a rule that appears relevant fails. For example, often sEBL tries to satisfy a rule that specifies moving four blocks to meet the goal of having a block at a given height, only to fail after much effort because all combinations of four blocks exceed the limitations on the tower height. On average, sEBL tries about $1\frac{1}{2}$ rules before solving a problem. When the effects of a rule are unrelated to the current goal, much less time will be wasted, especially if a complicated data structure is used to organize rules according to the goals they support.

Chapter 12

Conclusion

Most research in explanation-based learning involves relaxing constraints on the entities in a situation, rather than generalizing the number of entities themselves. Nonetheless, many important concepts require generalizing number. To acquire these concepts in an explanation-based fashion, the *structure* of the explanations must be generalized. Two theories of how this can be done have been presented. Implementations of both theories have been tested.

Physics 101 is a mathematical reasoning system, offered as a psychologically-plausible model, that performs explanation-based learning in mathematically-oriented domains. This system's understanding and generalization processes are guided by the manner in which variables are cancelled in a specific problem. Attention focusses on how *obstacles* are eliminated in the specific problem. Obstacles are variables that preclude the direct evaluation of the unknown. Cancelling these variables allows the determination of the value of the unknown. One important feature of analyzing variable cancellations is that the generalization of number is motivated. The explanation of a specific calculation closely guides the construction of a general version of the calculation, from which a new general concept is extracted.

The **BAGGER** system is a domain-independent approach to the problem of generalizing the structure of explanations. The approach relies on a shift in representation which accommodates indefinite numbers of rule applications. This system analyzes explanation structures and detects repeated, inter-dependent applications of rules. Once the system finds a rule on which to focus attention, it determines how an *arbitrary* number of instantiations of this rule can be concatenated together. This indefinite-length collection of rules is conceptually merged into the explanation, replacing the specific-length collection of rules, and an extension of a standard explanation-based algorithm produces a new rule from the augmented explanation.

The major contributions of this research are:

- (1) The claim that explanations that suffice to understand a specific example are not always sufficient to directly produce the complete underlying general concept. Often the structure of the explanation must be generalized. This is the central theme running throughout this thesis. Two implemented systems, tested on a number of problems from several domains and a large scale empirical investigation, substantiate this claim.
- (2) The idea that mathematical calculations can be understood and properly generalized by focussing on the idea of *obstacles* and their cancellation. This hypothesis is successfully tested in the **Physics 101** system. It is claimed that reasoning about mathematical cancellations allows the constraints inherent in mathematically-based domains to become apparent. One such example is the concept of momentum conservation. The construct of a *cancellation graph* is offered as a data structure for representing obstacle cancellations and forms the backbone of **Physics 101**'s understanding and generalization algorithms.
- (3) An alternate notion of explanation-based generalization. In **Physics 101** the explanation of a specific solution closely guides the *reconstruction* of the solution in the general case. Other generalization algorithms directly use the specific explanation. During **Physics 101**'s construction of the general calculation no problem-solving search is performed — construction follows deterministically from the specific calculation, leading to a relatively efficient process. The new calculation is often substantially more general, in terms of its structure as well as its variables, than the specific calculation. The number of entities and the identity of the operations performed may be generalized.

- (4) The notion of a *special case* as a way to address the operationality/generality problem. By properly organizing acquired concepts, the advantages of both operational and general rules can be obtained, while minimizing the disadvantages of both.
- (5) A domain-independent approach to the problem of generalizing to N . The **BAGGER** algorithm generalizes the structure of explanations so that an indefinite number of rule applications are supported. This algorithm is particularly applicable to domains expressed using situation calculus. The concept of *unwindable* rules supports the goal of expressing the preconditions of acquired rules only in terms of the initial state. The idea of a *rule instantiation sequence* (RIS) is presented as a data structure that accommodates an arbitrary number of applications of a specific rule. Specific explanations are reformulated in terms of this rule instantiation sequence.
- (6) An empirical demonstration of the value of generalizing number in particular and of explanation-based learning in general. The experiments performed provide no evidence for the claim that learning can have an overall detrimental effect. Although some problems will be solved more slowly after learning, this negative effect is overwhelmed by the speed-ups afforded by learning on the majority of problems.
- (7) An empirical demonstration of the advantages of learning by observing the intelligent behavior of external agents. The high cost a system must frequently pay if it solves all of its problems on its own can dissipate much of the gains it achieves from learning. A substantial improvement can be achieved by initially observing an expert solve several sample problems. No special ordering of the examples is necessary.

12.1. Relation to Other Work

Besides **Physics 101** and **BAGGER**, several other explanation-based approaches to generalizing the structure of explanations have been recently proposed elsewhere. This section reviews these approaches. Additional related work is discussed in the following section, where several open research issues are presented. Also, additional work on learning in mathematically-based domains is reviewed in section 4.6.

Prieditis

Prieditis [Prieditis86] has developed a system which learns macro-operators representing sequences of repeated STRIPS-like operators. His approach analyzes the constraints imposed by the connections of the precondition, *ADD*, and *DELETE* lists of the operators determined to be of interest. This produces an iterative macro-operator that accommodates an indefinite number of repeated operator inter-connections.

While **BAGGER** is very much in the spirit of Prieditis' work, STRIPS-like operators impose unwarranted restrictions (see section 8.2). For instance, **BAGGER**'s use of predicate calculus allows generalization of repeated structure and repeated actions in a uniform manner. In addition, the **BAGGER** approach accommodates the use of additional inference rules to reason about what is true in a state. Everything need not appear explicitly in the focus rule. For example, in the stacking example, other rules are used to determine the height of a tower and that an object is clear when the only object it supports is transferred. Also, instantiations of the focus rule do not have to connect directly — intervening inference rules can be involved when determining that the results of one instantiation partially support the preconditions of another. Finally, there is nothing in Prieditis' approach that corresponds to **BAGGER**'s unwinding operation, nor are disjunctive rules learned.

Cheng and Carbonell

In the **FERMI** system [Cheng86], cyclic patterns are recognized using empirical methods and the detected repeated pattern is generalized using explanation-based learning techniques. Unlike the other systems for number generalization, cyclic patterns are detected by analyzing changes in subgoals. The cost of this is explained in the next paragraph. Except for **Physics 101**, which analyzes obstacle cancellations, the other number generalization systems focus directly on rule applications in order to decide where to generalize number. A major strength of the **FERMI** system is the incorporation of conditionals within the learned macro-operator, allowing this system to learn disjunctive rules.

However, unlike the techniques implemented in **BAGGER** and **Physics 101**, the rules acquired by **FERMI** are not fully based on an explanation-based analysis of an example, and so are not guaranteed to always work. For example, **FERMI** learns a strategy for solving a set of linear algebraic equations. None of the preconditions of the strategy check that the equations

are linearly independent. The learned strategy will appear applicable to the problem of determining x and y from the equations $3x + y = 5$ and $6x + 2y = 10$. After a significant amount of work, the strategy will terminate unsuccessfully. This is one of the costs of focussing on changes at the subgoal level. Since several rules can cause the same subgoal change, the problem of analyzing the interactions between successive cycles is greatly complicated. A flexible agenda-based problem solver is used to circumvent this complication.

Cohen

Cohen [Cohen87] has recently developed and formalized another approach to the problem of generalizing number. His system generalizes number by constructing a finite-state control mechanism that deterministically directs the construction of proofs similar to the one used to justify the specific example. This is accomplished by first producing a linear *proof automaton*, a variant of a finite state automaton, that produces only one proof — that of the specific example. This proof automaton is then repeatedly reduced by merging states according to the constraints imposed by the inference rules used. Each reduction produces a loop in the automaton, while still keeping the automaton deterministic. Because the final automaton is deterministic, no search need be performed when applying it to future problems. One significant property of his method is that it can generate proof procedures involving tree traversals and nested loops.

A major difference between Cohen's method and other explanation-based algorithms is that in his approach no "internal nodes" of the explanation are eliminated during generalization. In other explanation-based algorithms, only the leaves of the operationalized explanation appear in the acquired rule. The generalization process guarantees that all of the inference rules within the explanation apply in the general case, and the final result can be viewed as a compilation of the effect of combining these rules as generally as possible. Hence, to apply the new rule, only the general versions of these leave nodes need be satisfied. In Cohen's approach, every inference rule used in the original explanation is explicitly incorporated into the final result. Each rule may again be applied when satisfying the acquired rule. Hence, there is nothing in this approach corresponding to unwinding a rule from an arbitrary state back to the initial state, and the efficiency gains obtained by doing this are not achieved. Finally, because the final automaton is deterministic, it incorporates disjunctions only in a limited way. For example, if at some point two choices are equally general, the ordering in the final rule will be the same as that seen in the specific example.

Mooney

Another aspect of generalizing the structure of explanations involves generalizing the *organization* of the nodes in the explanation, rather than generalizing the *number* of nodes. An approach of this form is presented in [Mooney88], where the temporal order of actions is generalized in plan-based explanations. The approach is limited to domains expressed in the STRIPS-formalism [Fikes71].

Related Work in Similarity-Based Learning

The problem of generalizing to N has also been addressed within the paradigms of similarity-based learning [Andreae84, Dietterich83, Dufay84, Holder88, Michalski83, Whitehall87, Wolff82] and automatic programming [Biermann78, Kodratoff79, Siklossy75, Summers77]. A general specification of number generalization has been advanced in [Michalski83]. Michalski proposes a set of generalization rules including a *closing interval rule* and several *counting arguments rules* which can generate number-generalized structures. The difference between such similarity-based approaches and BAGGER's explanation-based approach is that the newly formed similarity-based concepts typically require verification from corroborating examples, whereas the explanation-based concepts are immediately supported by the domain theory.

12.2. Some Open Research Issues

The Physics 101 and BAGGER systems have taken important steps towards the solution to the problem of generalizing the structure of explanations. However, the research is still incomplete. From the vantage point of the current results, several avenues of future research are apparent. While there are many issues related to performing, understanding, and generalizing mathematical calculations, this section only addresses issues related to the problem of generalizing explanation structures. Most of the discussion relates to the approach taken in BAGGER in addressing the generalizing to N problem.

The topics discussed in this section are:

- Detecting Fortuitous Circumstances — Deciding When to Learn
- Choosing a Focus Rule

- Representing Preconditions in Terms of Sets Rather than Sequences
- Guaranteeing Termination
- Interleaving Generalization to N
- Adding Extra Preconditions for Efficiency
- Efficiently Ordering Conjunctive Goals
- Satisfying Global Constraints
- Acquiring Accessory Inter-Situational Rules
- Solving Recurrences
- Empirically Analyzing EBL in a Domain-Independent Manner
- Handling Incomplete, Imperfect, and Intractable Domain Models

Detecting Fortuitous Circumstances — Deciding When to Learn

Possibly the major issue in generalizing the structure of explanations is that of deciding when there is enough information in the specific explanation to generalize its structure. Due to the finiteness of a specific problem, fortuitous circumstances in the specific situation may have allowed shortcuts in the solution. Complications in the general case may not have been faced. Hence the specific example provides no guidance as to how they should be addressed.

One aspect of this issue is recognizing the general problems inherent in the specific example. The notion of *obstacles* serves this role in **Physics 101**. The specific solution must eliminate all of the specific counterparts of the general versions of the obstacles. If a general obstacle has no counterpart in the specific example, nothing is learned. This is the reason that a two-ball collision does not provide enough information from which to learn the concept of momentum conservation (see section 7.4).

In **BAGGER**, the requirement that, for an application of a focus rule to be generalized, it be viewable as the arbitrary *i*th application also addresses the problem of recognizing fortuitous circumstances. If there is not enough information to view it as the *i*th application, it is likely that some important issue is not addressed in this focus rule application. However, more powerful techniques for detecting fortuitous circumstances need to be developed. One approach may be to abstract the notion of *obstacle* to cover more than mathematical cancellations.

An example in section 10.2 shows that **BAGGER** does not always acquire the fully general concept. Rather than learning that a cleared block remains clear as long as nothing else is placed

on it, from a particular example it only learns that a block is clear in the situation following the one in which the block on it is moved. In a more complicated example, **BAGGER** learns the more general concept. A second aspect of the issue of fortuitous circumstances is realizing that an acquired rule is overly-specific and should be refined or replaced. Besides refining the preconditions of an acquired rule as a result of later experiences, since **BAGGER** learns disjunctive rules it needs to be capable of adding disjuncts to existing rules, rather than storing multiple specializations of the same general rule. Determining which acquired rule "almost" solves a problem and, hence, is a candidate for refinement or replacement is a sizable problem. This is especially complicated when rules are insufficient because they are overly restrictive, rather than because they lead to incorrect results.

A learning system must insure that the rules it tries to apply to problems are likely to prove successful, otherwise the gains achieved by learning can be dissipated by the time spent attempting to apply rules that usually fail. This problem can be more severe when generalizing number because the specific explanation is generalized further than in standard explanation-based learning. There are three basic ways to address the problem of potentially being swamped by too many acquired rules [Fikes72]. One, care can be taken when deciding when to learn. Two, the collection of rules can be organized so that those most likely to be successful are tried first. Three, rules can be refined or replaced when they are found to be insufficient. The first two approaches have been investigated in the **Physics 101** and **BAGGER** systems. All three merit additional research.

Choosing a Focus Rule

Also related to the issue of deciding when to learn, **BAGGER's** method of choosing a rule on which to focus attention should be improved. Currently the first detected instance of interconnected applications of a rule is used as the focus rule. However, there could be several occurrences that satisfy these requirements. Techniques for comparing alternative focus rules are needed. Inductive inference approaches to detecting repeated structures [Andreae84, Dietterich83, Dufay84, Holder88, Weld86, Whitehall87, Wolff82] may be applicable to the generation of candidate focus rules, from which the explanation-based capabilities of **BAGGER** can build. It may also be necessary to rearrange the nodes in an explanation in order for it to be in a form where the **BAGGER** algorithm can apply.

BAGGER uses the heuristic of requiring multiple instantiations of a rule in order for it to be a focus rule. The same heuristic is used when deciding if an instantiation should be viewed as an unwindable rule (section 9.1). It is possible to generalize number on the basis of one application of a rule [Cohen87 (appendix B), Shavlik87c]. However, it is more intuitively-pleasing that multiple instantiations are needed to trigger the extension, thereby increasing the chance that the acquired rule will prove frequently applicable in the future.

Deciding when to repeatedly apply a rule is more directly motivated in **Physics 101**. The goal of cancelling all of the general obstacles determines when a rule (such as Newton's third law) is to be repeatedly applied. While deciding when to learn is more directed in this system, the approach is specific to mathematical calculations.

Representing Preconditions in Terms of Sets Rather than Sequences

Investigating the generalization of operator application orderings within learned rules is another opportunity for future research. Currently, in the rules learned by the **BAGGER** algorithm, the order interdependence among rule applications is specified in terms of sequences of vectors. However, this is unnecessarily constraining. When valid, these constraints should be specified in terms of *sets* or *bags*¹ of vectors. This could be accomplished by reasoning about the semantics of the system's predicate calculus functions and predicates. Properties such as symmetry, transitivity, and reflexivity may help determine constraints on order independence [Shavlik87c]. Mooney [Mooney88 (chapter 6)] presents an algorithm for generalizing the order of a fixed number of actions in a plan expressed in the **STRIPS** formalism.

If a set satisfies a learned rule's antecedents, then *any* sequence derived from that set suffices. Conversely, if the vectors in a set fail to satisfy a rule's antecedents, there is no need to test each permutation of the elements. Unfortunately, testing all permutations occurs if the antecedents are unnecessarily expressed in terms of sequences. For example, assume the task at hand is to find enough heavy rocks in a storehouse to serve as ballast for a ship. A sequential rule may first add the weights in some order, find out that the sum weight of all the rocks in the room is insufficient, and then try another ordering for adding the weights. A rule specified in terms of sets would terminate after adding the weights once.

¹ A bag (or multi-set) is an *unordered* collection of elements in which an element can appear more than once.

Guaranteeing Termination

One important aspect of generalizing number is that the acquired rules may produce data structures whose size can grow without bound (for example, the rule instantiation sequence in **BAGGER**) or the algorithms that satisfy these rules may fall into infinite loops [Cohen87]. This means the issue of termination is significant. Although the *halting problem* is undecidable in general, in restricted circumstances termination can be proved [Manna74]. Techniques for proving termination need to be incorporated into systems that generalize number. A practical, but less appealing, solution is to place resource bounds on the algorithms that apply number-generalized rules [Cohen87], potentially excluding successful applications.

Interleaving Generalization to N

Performing multiple generalizations to N in a single problem is another research topic. Especially interesting is *interleaved generalization to N* . Here, in the final result, each application in an arbitrary length sequence would be supported by another sequence of arbitrary length. In other words, a portion of the intermediate antecedent of a **BAGGER** rule would be the antecedents of another **BAGGER** sequential rule. An examples where this is needed appears in section 10.3, where a plan for moving *two* towers, each containing any number of blocks, is learned. What should be learned is a plan for moving *any* number of such towers.

Learning an interleaved sequential rule from one example may be too ambitious. A more reasonable approach may be to first learn a simple sequential rule, and then use it in the explanation of a later problem, something the current implementation of **BAGGER** does not do. Managing the interactions between the two *RIS*'s is a major issue. See [Cohen87] for a promising approach to the problem of interleaved generalization to N .

Adding Extra Preconditions for Efficiency

Section 10.4 demonstrates the value of adding extra terms to the preconditions of sequential rules. Analyzing the properties of the predicates in the preconditions can lead to the insertion of additional preconditions that prevent unfruitful attempts to extend rule instantiation sequences. Adding extra preconditions can also support proofs of termination. For example, knowing that block heights are always positive can terminate the extension of the *RIS* when the size of the tower being constructed exceeds the upper bound on the goal height.

Domain-independent ways of analyzing sequential rules and adding such additional preconditions need to be developed.

Efficiently Ordering Conjunctive Goals

An additional area of future research involves investigating the most efficient ordering of conjunctive goals. Consider an acquired sequential rule which builds towers of a desired height, subject to the constraint that no block can be placed upon a narrower block. The goal of building such towers is conjunctive: the correct height must be achieved and the width of the stacked blocks must be monotonically non-increasing. The optimal ordering is to select the blocks subject only to the height requirement and then sort them by size to determine their position in the tower. The reason this works is that a non-increasing ordering of widths on any set of blocks is guaranteed so that no additional block-selection constraints are imposed by this conjunct. The system should ultimately detect and exploit this kind of decomposability to improve the efficiency of the new rules. Work on ordering conjunctive goals appears in [Smith85, Treitel86].

Satisfying Global Constraints

Satisfying global constraints poses an additional research problem. The sequential rules investigated in this paper are all *incremental* in that successive operator applications converge toward the goal achievement. This is not necessarily the case for all sequential rules. Consider a sequential rule for unstacking complex block structures subject to the global constraint that the partially dismantled structure always be stable. Removing one block can drastically alter the significance of another block with respect to the structure's stability. For some structures, only the subterfuge of adding a temporary support block or a counter-balance will allow unstacking to proceed. A block may be safe to remove at one point but be essential to the overall structural stability at the next, even though the block actually removed was physically distant from it. Such *non-incremental* effects are difficult to capture in sequential rules without permitting intermediate problem-solving within the rule execution.

The *RIS*, besides recording the focus rule's variable bindings, is used to store intermediate calculations, such as the height of the tower currently planned. Satisfying global constraints may require that the information in an *RIS* vector increase as the sequence lengthens. For example, assume that each block to be added to a tower can only support some block dependent

weight. The *RIS* may have to record the projected weight on each block while **BAGGER** plans the construction of a tower. Hence, as the sequence lengthens, each successive vector in the *RIS* will have to record information for one additional block. Figuratively speaking, the *RIS* will be getting longer and wider. The current **BAGGER** algorithm does not support this.

Acquiring Accessory Inter-Situational Rules

BAGGER and other such systems need to acquire accessory inter-situational rules, such as frame axioms, to complement their composite rules. Currently, each of **BAGGER**'s new sequential inference rules specifies how to achieve a goal involving some arbitrary aggregation of objects by applying some number of operators. These rules are useful in directly achieving goals that match the consequent, but do not effectively improve **BAGGER**'s back-chaining problem-solving ability. This is because currently **BAGGER** does not construct new frame axioms for the rules it learns. (This problem is not specific to generalizing to *N*. Standard explanation-based learning algorithms must also face it when dealing with situation calculus.)

There are several methods of acquiring accessory rules. They can be constructed directly by combining the accessory rules of operators that make up the sequential rule. This may be intractable as the number of accessory rules for initial operators may be large and they may increase combinatorially in sequential rules. Another, potentially more attractive, approach is to treat the domain theory, augmented by sequential rules, as intractable. Since the accessory rules for learned rules are derivable from existing knowledge of initial operators, the approach in [Chien87] might be used to acquire the unstated but derivable accessory rules when they are needed.

Solving Recurrences

Often a repeated process has a closed form solution. For example, summing the first *N* integers produces $\frac{N(N+1)}{2}$. There is no need to compute the intermediate partial summations. A *recurrence relation* is a recursive method for computing a sequence of numbers [Liu68].

Many recurrences can be solved to produce efficient ways to determine the *n*:th result in a sequence. It is this property that motivates the requirement that **BAGGER**'s preconditions be expressed solely in terms of the initial state. However, the rule instantiation sequence still holds intermediate results. While often this information is needed (if, for instance, the

resulting sequence of actions is to be executed in the external world), BAGGER would be more efficient if it could produce, whenever possible, number-generalized rules that did not require the construction of an *RIS*. If BAGGER observes the summation of, say, four numbers it will not produce the efficient result mentioned above. Instead it will produce a rule that stores the intermediate summations in the *RIS*. One extension that could be attempted is to create a library of templates for soluble recurrences, matching them against explanations. A more direct approach would be more appealing. Weld's [Weld86] technique of *aggregation* may be a fruitful approach. Aggregation is an abstraction technique for creating a continuous description from a series of discrete events.

Empirically Analyzing EBL in a Domain-Independent Manner

The empirical analysis presented in chapter 11 only involves the blocks world domain. While providing a major test of standard EBL and BAGGER under a variety of conditions, it is not clear how much the results depend on the peculiarities of that domain. A domain-independent analysis is needed.

One approach would be to randomly generate rules, initial states, and goals. Since an underlying assumption of EBL is that past problems are indicative of future problems, and hence the generalizations of their solutions are worth saving, with excessive randomness it is unlikely that learning will prove beneficial. However, varying such things as the ratio of the number of different predicates to the number of domain rules may provide an indication of the domains in which standard explanation-based learning and BAGGER perform well. Other properties to vary include the average number of antecedents in a rule, the number of different consequents, the probability that a predicate in the antecedents also appears in the consequents, and the probability that a variable appears in more than one antecedent of a rule. To complement empirical analyses of EBL, theoretical analyses are needed, as has been done for similarity-based learning [Angluin83, Gold67, Haussler87, Kearns87, Natarajan87, Valiant84]. An approach that applies Valiant's learning framework [Valiant84] to an aspect of explanation-based learning is described in [Mahadevan87].

Handling Incomplete, Imperfect, and Intractable Domain Models

Finally, it is important to investigate the issue of generalizing the structure of explanations in the context of incomplete, imperfect, and intractable domain models [Mitchell86, Rajamoney87]. In any real-world domain, a computer system's model can only approximate reality. Needed inference rules may be missing, while many of those possessed will lead to incorrect results in some situations. Furthermore, the complexity of problem solving prohibits any semblance of completeness. Thus far **BAGGER**'s sequential rules have relied on a correct domain model, and issues of intractability have not been addressed, other than the use of an outside expert to provide sample solutions when the construction of solutions from first principles is intractable. In **Physics 101**, reasoning about intractability and incompleteness could be incorporated by assuming obstacles are *negligible* and, hence, can be ignored (see section 3.3).

Explanation-based approaches to handling the problem of intractability appear in [Bennett87, Chien87, Doyle86, Gupta87, Hammond87, Mostow87, Tadepalli86]. A popular approach is to make simplifying assumptions when initially learning, then attempt to correct these assumptions if they later lead to failure. Work addressing issues related to imperfect and incomplete domain theories is presented in [Carbonell87, Falkenhainer87, Mahadevan87, Rajamoney85]. These issues can also be addressed by combining explanation-based techniques with other approaches to machine learning [Anderson87, Lebowitz86, Pazzani87, Porter86, Sims87]. Reducing the dependence of explanation-based learning on the assumption of tractable, complete, and correct domain theories is currently one of the most active areas in EBL. The techniques proposed must be extendible to handle the important problem of generalizing the structure of explanations.

12.3. Final Summary

This thesis opens with the claim that current approaches to explanation-based learning have a fundamental shortcoming — they do not alter the structure of their explanations. After motivating the need for generalizing the structure of explanations, two approaches that address this problem are presented.

The **Physics 101** system is presented first. Its learning algorithm is based on the idea that obstacles and their cancellations are fundamental to problem solving and learning in

mathematically-based domains. Mathematically-based domains are an area where the strengths of explanation-based learning are particularly appropriate, because explanation-based learning supports the construction of large concepts by analyzing how smaller concepts can be pieced together to solve a specific problem. Combining small concepts to form larger ones is the basis of progress in mathematical domains.

The processes of explanation construction, generalization, and problem solving in **Physics 101** are discussed, providing a view of a complete learning and problem-solving system. Organization of acquired concepts is discussed. In particular, the idea of storing special cases along with general concepts is presented. The acquisition of concepts from classical mechanics are used to illustrate the operation of **Physics 101**. These concepts cannot be properly acquired using standard explanation-based algorithms. The details of **Physics 101**'s algorithms are described and other approaches to learning in mathematical domains are reviewed.

The domain-independent **BAGGER** system is presented second. This is also a complete system that solves problems and learns. It is based on the properties of the syntactic structure of explanations, rather than semantic properties of the domain as in **Physics 101**, a system based on the semantics of obstacle cancellation. A range of problems that **BAGGER** properly generalizes are introduced, followed by an exposition of the its generalization algorithm. Several examples are then presented in detail, illustrating the strengths and weaknesses of the algorithm.

Following the discussion of the **BAGGER** generalization algorithm, a series of large-scale computer experiments are presented. The **BAGGER** system is compared to an implementation of a standard explanation-based generalization algorithm and to a problem-solving system that does no learning. Several issues in explanation-based learning are investigated, and information relevant to making decisions when designing an explanation-based learning system is reported. The results demonstrate the value of generalizing to N in particular and of explanation-based learning in general.

Finally, in this chapter, the important contributions of this work are reviewed. Following that, related approaches to the problem of generalizing the structure of explanations are presented. Several open research issues are then discussed.

Generalizing the structure of explanations is an important property currently lacking in most explanation-based systems. However, many concepts cannot be properly captured unless explanation-based learning systems possess this property. This research contributes to the theory and practice of explanation-based learning by developing and testing methods for extending the structure of explanations during generalization. It brings this field of machine learning closer to its goal of being able to acquire the full concept inherent in the solution to a specific problem.

Appendix A

BAGGER's Initial Inference Rules

The inference rules and axioms used in the **BAGGER** system are presented in this appendix. In these rules, all variables contain a leading "?" and are universally quantified. Conjunction is implicit. The construct $\{?a \mid ?b\}$ matches a list with head $?a$ and tail $?b$. For example, if matched with $\{x,y,z\}$, variable $?a$ is bound to x and $?b$ to $\{y,z\}$.

The first table contains those rules that describe properties of collections of objects. Included are rules for membership, rules for specifying how to add and remove elements, a rule about inequality, and rules about cardinality (size). These rules are used in the blocks world examples and in the example involving the setting of a table.

The second table contains *intra*-situation rules for the blocks world examples. For example, in the blocks world, there are tables and blocks. There are two types of blocks: boxes, which have flat tops, and pyramids. Rules for inferring that free space is present, a block is liftable, and an object is clear are included.

Table A.1 Rules for Collections of Objects

Rule	Description
$\text{Member}(\text{?x}, \text{?bag})$ \rightarrow $\text{Member}(\text{?x}, \{\text{?y} \mid \text{?bag}\})$	If an object is a member of a collection of objects, it is also a member of a superset of that collection.
$\text{Member}(\text{?x}, \{\text{?x} \mid \text{?bag}\})$	See if an object is in a collection of objects.
$\text{?x} \neq \text{?y}$ $\text{NotMember}(\text{?x}, \text{?bag})$ \rightarrow $\text{NotMember}(\text{?x}, \{\text{?y} \mid \text{?bag}\})$	If two objects are distinct, and the first is not in a collection of objects, then the first is not a member of the collection that results from adding the second object to the original collection.
$\text{NotMember}(\text{?x}, \phi)$	Nothing is a member of the empty set.
$\text{RemoveFromBag}(\text{?x}, \text{?bag1}, \text{?bag2})$ \rightarrow $\text{RemoveFromBag}(\text{?x}, \{\text{?y} \mid \text{?bag1}\}, \{\text{?y} \mid \text{?bag2}\})$	If two collections of objects are related by the removal of one object, then this same relation holds if a second object is added to each collection.
$\text{RemoveFromBag}(\text{?x}, \{\text{?x} \mid \text{?bag}\}, \text{?bag})$	Remove this object from a collection of objects, producing a new collection of objects.
$\text{AddToBag}(\text{?x}, \text{?bag1}, \text{?bag2})$ \rightarrow $\text{AddToBag}(\text{?x}, \{\text{?y} \mid \text{?bag1}\}, \{\text{?y} \mid \text{?bag2}\})$	If two collections of objects are related by the addition of one object, then this same relation holds if a second object is added to each collection.
$\text{AddToBag}(\text{?x}, \text{?bag}, \{\text{?x} \mid \text{?bag}\})$	Add this object to a collection of objects, producing a new collection of objects.
$\text{Size}(\text{?bag}, \text{?m})$ $\text{?n} = (\text{?m} + 1) \quad \rightarrow \quad \text{Size}(\{\text{?x} \mid \text{?bag}\}, \text{?n})$	The size of a collection of objects is increased by one if another object is added to the collection.
$\text{Size}(\phi, 0)$	The empty set has size zero.
$\text{?x} \neq \text{?y} \quad \rightarrow \quad \text{?y} \neq \text{?x}$	Inequality is reflexive.

Table A.2 Blocks World Intra-Situation Rules		
Rule		Description
Clear(?x,?s) FlatTop(?x)	→ FreeSpace(?x,?s)	If an object is clear and has a flat top, space is available.
Table(?x)	→ FreeSpace(?x,?s)	There always is room on a table.
Clear(?x,?s) Block(?x)	→ Liftable(?x,?s)	A block is liftable if it is clear.
Box(?x)	→ FlatTop(?x)	Boxes have flat tops.
Table(?x)	→ FlatTop(?x)	Tables have flat tops.
Box(?x)	→ Block(?x)	Boxes are a type of block.
Pyramid(?x)	→ Block(?x)	Pyramids are a type of block.
Supports(?x,φ,?s)	→ Clear(?x,?s)	An object is clear if it is supporting nothing.

The next two tables describe *inter*-situation inferences for the blocks world. The first rule in the table A.3 is the definition of the transfer action used in the examples. The other accessory rules describe how the location of blocks change after a transfer. Table A.3 contains rules for reasoning about support relationships following a transfer.

Table A.3 Some Blocks World Inter-Situation Rules

Rule	Description
<p>AchievableState(?s) Liftable(?x,?s) FreeSpace(?y,?s) $?x \neq ?y$ \rightarrow AchievableState(Do(Transfer(?x,?y),?s)) Clear(?x,Do(Transfer(?x,?y),?s)) On(?x,?y,Do(Transfer(?x,?y),?s))</p>	<p>If the top of an object is clear in some achievable state and there is free space on another object, then the first object can be moved from its present location to the new location. However, an object cannot be moved onto itself. Moving creates a new state in which the moved object is still clear but (possibly) at a new location.</p>
<p>Xpos(?y,?xpos,?s) \rightarrow Xpos(?x,?xpos,Do(Transfer(?x,?y),?s))</p>	<p>After a transfer, the object moved is centered (in the X-direction) on the object upon which it is placed.</p>
<p>$?x \neq ?y$ Height(?x,?hx) Ypos(?y,?ypos,?s) $?ypos2 = (?hx + ?ypos)$ \rightarrow Ypos(?x,?ypos2,Do(Transfer(?x,?y),?s))</p>	<p>After a transfer, the Y-position of the object moved is determined by adding its height to the Y-position of the object upon which it is placed.</p>
<p>$?a \neq ?x$ Xpos(?a,?xpos,?s) \rightarrow Xpos(?a,?xpos,Do(Transfer(?x,?y),?s))</p>	<p>All blocks, other than the one moved, remain in the same X-position after a transfer.</p>
<p>$?a \neq ?x$ Ypos(?a,?xpos,?s) \rightarrow Ypos(?a,?xpos,Do(Transfer(?x,?y),?s))</p>	<p>All blocks, other than the one moved, remain in the same Y-position after a transfer.</p>

Table A.4 Inter-Situation Rules about Block Support

Rule	Description
$?a \neq ?x$ $On(?a, b, ?s)$ \rightarrow $On(?a, ?b, Do(Transfer(?x, ?y), ?s))$	If an object is not moved, it remains on the same object.
$?u \neq ?y$ $Supports(?u, ?items, ?s)$ $NotMember(?x, ?items)$ \rightarrow $Supports(?u, ?items, Do(Transfer(?x, ?y), ?s))$	If an object neither supports the moved object before the transfer, nor is the new supporter, then the collection of objects it supports remains unchanged.
$?u \neq ?y$ $Supports(?u, ?items, ?s)$ $Member(?x, ?items)$ $RemoveFromBag(?x, ?items, ?new)$ \rightarrow $Supports(?u, ?new, Do(Transfer(?x, ?y), ?s))$	If an object is not the new support of the moved object, but supported it before the transfer, then the moved object must be removed from the collection of objects being supported.
$Supports(?y, ?items, ?s)$ $NotMember(?x, ?items)$ $AddToBag(?x, ?items, ?new)$ \rightarrow $Supports(?y, ?new, Do(Transfer(?x, ?y), ?s))$	After the transfer, another item is supported if the new supporter did not previously support the moved object.
$Supports(?y, ?items, ?s)$ $Member(?x, ?items)$ \rightarrow $Supports(?y, ?items, Do(Transfer(?x, ?y), ?s))$	If the moved object is transferred to the object that previously supported it, the new supporter still supports the same objects.

Table A.5 presents the two rules used in the example about setting tables. The last table, table A.6, contains the rules used in the circuit design example involving DeMorgan's law.

Table A.5 Rules about Setting Tables	
Rule	Description
Table(?tbl) AchievableState(?s) AvailUtensilSets(?utenSets,?s) Member(?utens,?utenSets) RemoveFromBag(?utens,?utenSets,?newUsets) ClearTableLocs(?tbl,?locs,?s) Member(?loc,?locs) RemoveFromBag(?loc,?locs,?newLocs) → AchievableState(Do(SetTblLoc(?tbl,?loc,?utens))) ClearTableLocs(?tbl,?newLocs,Do(SetTblLoc(?tbl,?loc,?utens))) AvailUtensilSets(?newUsets,Do(SetTblLoc(?tbl,?loc,?utens)))	To set a place on a table, there must be an available set of utensils and an open location on the table. Following the settings, the used resources are no longer available.
Table(?tbl) TableSize(?tbl,?size) ClearTableLocs(?tbl,?locs,?s) Size(?locs,?n) ?size = (?space + ?n) → SetTableLocations(?tbl,?spaces,?s)	The number of tables locations that are set is determined by subtracting the number that are clear from the table capacity.

Table A.6 Circuit Design Rules	
Rule	Description
$\text{Wire}(\text{?wire})$ \rightarrow $\text{Implements}(\text{?wire}, \text{?wire})$	Wires are easily implemented.
$\text{Implements}((\neg \text{?x} \wedge \neg \text{?y}), \text{?a})$ \rightarrow $\text{Implements}(\neg(\text{?x} \vee \text{?y}), \text{?a})$	DeMorgan's law for two-input gates.
$\text{Implements}(\text{?x}, \text{?a})$ $\text{Implements}(\text{?y}, \text{?b})$ \rightarrow $\text{Implements}((\text{?x} \wedge \text{?y}), (\text{?a} \wedge \text{?b}))$	An implementation of a circuit headed by a two-input AND can be found by finding implementations of the two sub-circuits.
$\text{Implements}(\text{?x}, \text{?y})$ \rightarrow $\text{Implements}(\neg \neg \text{?x}, \text{?y})$	Double negations have no effect.

Appendix B

Standard Deviations of Experimental Results

The means and standard deviations of several experimental measurements are reported in this appendix, in order to provide an indication of the variability of the results. The standard deviation (σ) is defined as follows, where X contains the collection of results from the 25 experimental runs.

$$\sigma(X) = \sqrt{\text{mean}(X^2) - \text{mean}^2(X)}$$

The standard deviation is the positive square root of the difference between the mean of the square of each result and the square of the mean result.

Table B.1 presents the solution time data for a variety of experiments. Data in the autonomous mode is averaged over problems 26-50, while in the training mode the 20 problems in the training set are used. Data pertaining to the number of rules learned appears in table B.2. Unless otherwise stated, the results are for building towers.

Table B.1 Means and Standard Deviations for Various Experiments

<i>Description</i>	<i>Operationality</i>		<i>Generality</i>	
	<i>Mean</i>	<i>Std. Dev.</i>	<i>Mean</i>	<i>Std. Dev.</i>
Mean Solution Time				
autonomous mode				
<i>No-Learn</i>	64,500 sec	36,500	79,300	26,900
<i>Std-EBL</i>	6790	9,140	8100	11,500
<i>BAGGER</i>				
w/o std-EBL	6660	10,200	3720	7,320
with std-EBL	3750	9,000	57.5	96.3
autonomous mode (external solutions after 10,000 sec)				
<i>Std-EBL</i>	318	239	1290	498
<i>BAGGER</i>				
w/o std-EBL	197	251	133	171
with std-EBL	145	267	52.1	72.0
autonomous mode (external solutions if all rules fail)				
<i>Std-EBL</i>	133	37.4	1470	714
<i>BAGGER</i>				
w/o std-EBL	28.6	22.3	40.5	8.9
with std-EBL	21.9	18.8	36.3	6.2
training mode				
<i>No-Learn</i>	91,000	36,900	68,400	38,600
<i>Std-EBL</i>	155	322	780	3,580
<i>BAGGER</i>				
w/o std-EBL	26.6	81.5	35.6	84.1
with std-EBL	na	na	46.2	162
training mode (clearing)				
<i>No-Learn</i>	na	na	4,200,000	2,650,000
<i>Std-EBL</i>	na	na	57.9	132
<i>BAGGER</i>				
w/o std-EBL	na	na	15.4	20.5

na - not available (experiment not run)

Table B.2 Standard Deviations of Rules Learned for Various Experiments				
<i>Description</i>	<i>Operationality</i>		<i>Generality</i>	
	<i>Mean</i>	<i>Std Dev</i>	<i>Mean</i>	<i>Std. Dev.</i>
Rules Learned				
autonomous mode				
<i>Std-EBL</i>	5.52	1.09	4.28	1.00
<i>BAGGER</i>				
w/o std-EBL	2.60	0.75	1.72	0.45
with std-EBL	3.56	0.98	1.96	0.74
training mode				
<i>Std-EBL</i>	6.88	1.31	5.96	1.11
<i>BAGGER</i>				
w/o std-EBL	3.24	0.95	2.01	0.75
with std-EBL	na	na	3.89	1.10
training mode (clearing)				
<i>Std-EBL</i>	na	na	3.96	0.68
<i>BAGGER</i>				
w/o std-EBL	na	na	1.36	0.53

na - not available (experiment not run)

References

- [Ahn87] W. Ahn, R. J. Mooney, W. F. Brewer and G. F. DeJong, "Schema Acquisition from One Example: Psychological Evidence for Explanation-Based Learning," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, July 1987.
- [Anderson83] J. R. Anderson, *The Architecture of Cognition*, Harvard University Press, Cambridge, MA, 1983.
- [Anderson87] J. R. Anderson, "Causal Analysis and Inductive Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 288-299.
- [Anderson88] J. R. Anderson and R. Thompson, "Use of Analogy in a Production System Architecture," in *Similarity and Analogical Reasoning*, S. Vosniadou and A. Ortony (ed.), Cambridge University Press, Cambridge, England, forthcoming.
- [Andreae84] P. M. Andreae, "Justified Generalization: Acquiring Procedures from Examples," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, January 1984. (Also appears as Technical Report 834, MIT AI Laboratory.)
- [Angluin83] D. Angluin and C. H. Smith, "Inductive Inference: Theory and Methods," *Computing Surveys* 15, 3 (1983), pp. 237-269.
- [Araya84] A. Araya, "Learning Problem Classes by Means of Experimentation and Generalization," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 11-15.
- [Benanav85] D. Benanav, D. Kapur and P. Narendran, "Complexity of Matching Problems," *Proceedings of the First International Conference on Rewriting Techniques and Applications*, Dijon, France, May 1985.

- [Bennett87] S. W. Bennett, "Approximation in Mathematical Domains," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 239-241.
- [Biermann78] A. W. Biermann, "The Inference of Regular LISP Programs from Examples," *IEEE Transactions on Systems, Man and Cybernetics* 8, 8 (1978), pp. 585-600.
- [Bransford76] J. D. Bransford and J. J. Franks, "Toward a Framework for Understanding Learning," in *The Psychology of Learning and Motivation, Volume 10*, G. H. Bower (ed.), Academic Press, New York, NY, 1976, pp. 93-127.
- [Bundy79] A. Bundy, L. Byrd, G. Luger, C. Mellish and M. Palmer, "Solving Mechanics Problems using Meta-Level Inference," in *Expert Systems in the Micro-Electronic Age*, D. Michie (ed.), Edinburgh University Press, Edinburgh, Scotland, 1979, pp. 50-64.
- [Bundy81] A. Bundy and B. Welham, "Using Meta-level Inference for Selective Application of Multiple Rewrite Rules in Algebraic Manipulation," *Artificial Intelligence* 16, 2 (1981), pp. 189-212.
- [Bundy83] A. Bundy, *The Computer Modelling of Mathematical Reasoning*, Academic Press, New York, NY, 1983.
- [Bundy85] A. Bundy, B. Silver and D. Plummer, "An Analytical Comparison of Some Rule Learning Programs," *Artificial Intelligence* 27, 2 (1985), pp. 137-181.
- [Carbonell87] J. G. Carbonell and Y. Gil, "Learning by Experimentation," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 256-266.
- [Chafe75] W. Chafe, "Some Thoughts on Schemata," *Theoretical Issues in Natural Language Processing 1*, Cambridge, MA, 1975, pp. 89-91.
- [Charniak76] E. Charniak, "A Framed PAINTING: The Representation of a Common Sense Knowledge Fragment," *Cognitive Science* 4, (1976), pp. 355-394.
- [Charniak80] E. Charniak, C. Riesbeck and D. McDermott, *Artificial Intelligence Programming*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1980.
- [Chase73] W. G. Chase and H. A. Simon, "Perception in Chess," *Cognitive Psychology* 4, (1973), pp. 55-81.
- [Cheng86] P. Cheng and J. G. Carbonell, "The FERMI System: Inducing Iterative Macro-operators from Experience," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 490-495.
- [Chi81] M. T. Chi, P. J. Feltovich and R. Glaser, "Categorization and Representation of Physics Problems by Experts and Novices," *Cognitive Science* 5, 2 (1981), pp. 121-152.
- [Chi82] M. Chi, R. Glaser and E. Rees, "Expertise in Problem Solving," in *Advances in the Psychology of Human Intelligence, Volume 1*, R. Sternberg (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1982, pp. 7-75.

- [Chien87] S. A. Chien, "Simplifications in Temporal Persistence: An Approach to the Intractable Domain Theory Problem in Explanation-Based Learning," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, September 1987. (Also appears as Technical Report UILU-ENG-87-2255, AI Research Group, Coordinated Science Laboratory.)
- [Cohen87] W. W. Cohen, "A Technique for Generalizing Number in Explanation-Based Learning," ML-TR-19, Department of Computer Science, Rutgers University, New Brunswick, NJ, September 1987.
- [Cullingford78] R. E. Cullingford, "Script Application: Computer Understanding of Newspaper Stories," Technical Report 116, Department of Computer Science, Yale University, New Haven, CT, January 1978.
- [DeJong81] G. F. DeJong, "Generalizations Based on Explanations," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August 1981, pp. 67-70.
- [DeJong83] G. F. DeJong, "Acquiring Schemata through Understanding and Generalizing Plans," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 462-464.
- [DeJong86] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning* 1, 2 (1986), pp. 145-176.
- [Dietterich82] T. G. Dietterich, B. London, K. Clarkson and G. Dromney, "Learning and Inductive Inference," in *The Handbook of Artificial Intelligence*, Vol. III, P. R. Cohen & E. A. Feigenbaum (ed.), William Kaufman, Inc., Los Altos, CA, 1982.
- [Dietterich83] T. G. Dietterich and R. S. Michalski, "A Comparative Review of Selected Methods for Learning from Examples," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 41-81.
- [Dietterich86] T. G. Dietterich, "Learning at the Knowledge Level," *Machine Learning* 1, 3 (1986), pp. 287-316.
- [Dijkstra76] E. W. Dijkstra, *A Discipline of Programming*, Prentice Hall, Englewood Cliffs, NJ, 1976.
- [Doyle86] R. Doyle, "Constructing and Refining Causal Explanations from an Inconsistent Domain Theory," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 538-544.
- [Dufay84] B. Dufay and J. Latombe, "An Approach to Automatic Robot Programming Based on Inductive Learning," in *Robotics Research: The First International Symposium*, MIT Press, Cambridge, MA, 1984, pp. 97-115.
- [Egan74] D. E. Egan and J. G. Greeno, "Theory of Rule Induction: Knowledge Acquisition in Concept Learning, Serial Pattern Learning and Problem Solving," in *Knowledge and Cognition*, L. W. Gregg (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1974.

- [Ellman85] T. Ellman, "Generalizing Logic Circuit Designs by Analyzing Proofs of Correctness," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 643-646.
- [Ellman87] T. Ellman, "Explanation-Based Learning: A Survey of Programs and Perspectives," Technical Report, Department of Computer Science, Columbia University, New York City, NY, May 1987.
- [Fahlman74] S. Fahlman, "A Planning System for Robot Construction Tasks," *Artificial Intelligence* 5, 1 (1974), pp. 1-49.
- [Falkenhainer86] B. C. Falkenhainer and R. S. Michalski, "Integrating Quantitative and Qualitative Discovery: The ABACUS System," *Machine Learning* 1, 4 (1986), pp. 367-401.
- [Falkenhainer87] B. C. Falkenhainer, "Scientific Theory Formation Through Analogical Inference," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 218-229.
- [Fateman85] R. J. Fateman, "Eleven Proofs of $\sin^2 x + \cos^2 x = 1$," *SIGSAM Bulletin* 19, 2 (1985), pp. 25-28.
- [Feigenbaum63] E. A. Feigenbaum, "The Simulation of Natural Learning Behavior," in *Computers and Thought*, E. A. Feigenbaum and J. Feldman (ed.), McGraw-Hill, New York, NY, 1963.
- [Fikes71] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence* 2, 3/4 (1971), pp. 189-208.
- [Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence* 3, 4 (1972), pp. 251-288.
- [Fikes75] R. Fikes, "Deductive Retrieval Mechanisms for State Description Models," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, U.S.S.R., August 1975, pp. 99-106.
- [Floyd67] R. W. Floyd, "Assigning Meanings to Programs," in *Mathematical Aspects of Computer Science*, J. T. Schwartz (ed.), American Mathematical Society, Providence, RI, 1967, pp. 19-32.
- [Forbus83] K. Forbus and D. Gentner, "Learning Physical Domains: Towards a Theoretical Framework," *Proceedings of the 1983 International Machine Learning Workshop*, Urbana, IL, June 1983, pp. 198-202.
- [Geddes82] K. Geddes, G. Gonnet and B. Char, "MAPLE User's Manual, Second Edition," Technical Report CS-82-40, University of Waterloo, Ontario, CANADA, December 1982.
- [Gentner88] D. Gentner, "Mechanisms of Analogical Learning," in *Similarity and Analogical Reasoning*, S. Vosniadou and A. Ortony (ed.), Cambridge University Press, Cambridge, England, forthcoming.
- [Gick83] M. I. Gick and K. I. Holyoak, "Schema Induction and Analogical Transfer," *Cognitive Psychology* 15, (1983), pp. 1-38.

- [Gold67] E. M. Gold, "Language Identification in the Limit," *Information and Control* 10, (1967), pp. 447-474.
- [Green69] C. C. Green, "Application of Theorem Proving to Problem Solving," *Proceedings of the First International Joint Conference on Artificial Intelligence*, Washington, D.C., August 1969, pp. 219-239.
- [Griener88] R. Griener, "Learning by Understanding Analogies," *Artificial Intelligence*, forthcoming.
- [Gupta87] A. Gupta, "Explanation-Based Failure Recovery," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 606-610.
- [Hammond87] K. J. Hammond, "Learning and Reusing Explanations," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 141-147.
- [Haussler87] D. Haussler, "Learning Conjunctive Concepts in Structural Domains," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 466-470.
- [Hearn84] A. Hearn, "REDUCE User's Manual, Version 3.1," Technical Report CP-78, The RAND Corporation, Santa Monica, CA, April 1984.
- [Hill87] W. L. Hill, "Machine Learning for Software Reuse," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 338-344.
- [Hinsley77] D. A. Hinsley, J. R. Hayes and H. A. Simon, "From Words to Equations: Meaning and Representation on Algebra Word Problems," in *Cognitive Processes in Comprehension*, P. A. Carpenter and M. A. Just (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1977, pp. 89-105.
- [Hinton86] G. E. Hinton and T. J. Sejnowski, "Learning and Relearning in Boltzmann Machines," in *Parallel Distributed Processing, Vol. I*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 282-317.
- [Hirsh87] H. Hirsh, "Explanation-Based Generalization in a Logic-Programming Environment," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 221-227.
- [Holder88] L. B. Holder, "Discovering Substructures in Examples," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, forthcoming.
- [Hunt66] E. B. Hunt, J. Marin and P. J. Stone, *Experiments in Induction*, Academic Press, New York, NY, 1966.
- [Iba85] G. A. Iba, "Learning by Discovering Macros in Puzzle Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 640-642.
- [Kearns87] M. Kearns, M. Li, L. Pitt and I. G. Valiant, "Recent Results on Boolean Concept Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 337-352.

- [Kedar-Cabelli87] S. T. Kedar-Cabelli and L. T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 383-389.
- [Keller87a] R. M. Keller, "The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance," Ph.D. Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1987. (Also appears as Technical Report ML-TR-7, Department of Computer Science, Rutgers University.)
- [Keller87b] R. M. Keller, "Defining Operationality for Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 482-487.
- [Keller87c] R. M. Keller, "Concept Learning in Context," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 91-102.
- [Kirkpatrick83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science* 220, (1983), pp. 671-680.
- [Kodratoff79] Y. Kodratoff, "A Class of Functions Synthesized from a Finite Number of Examples and a LISP Program Scheme," *International Journal of Computer and Information Sciences* 8, 6 (1979), pp. 489-521.
- [Kokar86] M. M. Kokar, "Determining Arguments of Invariant Functional Descriptions," *Machine Learning* 1, 4 (1986), pp. 403-422.
- [Kolodner84] J. L. Kolodner, *Retrieval and Organizational Strategies in Conceptual Memory*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1984.
- [Korf85] R. E. Korf, "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence* 27, 1 (1985), pp. 97-109.
- [Laird86] J. E. Laird, P. S. Rosenbloom and A. Newell, *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Kluwer Academic Publishers, Norwell, MA, 1986.
- [Langley81] P. Langley, "Data-Driven Discovery of Physical Laws," *Cognitive Science* 5, 1 (1981), pp. 31-54.
- [Langley87] P. Langley, H. A. Simon, G. L. Bradshaw and J. M. Zytkow, *Scientific Discovery: Computational Explorations of the Creative Processes*, MIT Press, Cambridge, MA, 1987.
- [Larkin80] J. H. Larkin, J. McDermott, D. P. Simon and H. A. Simon, "Models of Competence in Solving Physics Problems," *Cognitive Science* 4, 4 (1980), pp. 317-345.
- [Lebowitz80] M. Lebowitz, "Generalization and Memory in an Integrated Understanding System," Technical Report 186, Ph.D. Thesis, Department of Computer Science, Yale University, New Haven, CT, 1980.
- [Lebowitz86] M. Lebowitz, "Integrated Learning: Controlling Explanation," *Cognitive Science* 10, 2 (1986), pp. 219-240.

- [Lenat76] D. B. Lenat, "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," Ph.D. Thesis, Department of Computer Science, Stanford University, Stanford, CA, 1976.
- [Liu68] C. L. Liu, *Introduction to Combinatorial Mathematics*, Computer Science Press, McGraw Hill, New York, NY, 1968.
- [Mahadevan85] S. Mahadevan, "Verification-Based Learning: A Generalization Strategy for Inferring Problem-Reduction Methods," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 616-623.
- [Mahadevan87] S. Mahadevan and P. Tadepalli, "On the Tractability of Learning from Incomplete Theories," Working Paper 71, Department of Computer Science, Rutgers University, New Brunswick, NJ, October 1987.
- [Manna70] Z. Manna, "Termination of Programs Represented as Interpreted Graphs," *Proceedings of the Spring Joint Computer Conference*, 1970, pp. 83-89.
- [Manna74] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, New York, NY, 1974.
- [Mathlab83] Mathlab, "MACSYMA Reference Manual, Version Ten," Laboratory for Computer Science, MIT, Cambridge, MA, December 1983.
- [McCarthy63] J. McCarthy, "Situations, Actions, and Causal Laws," memorandum, Stanford University, Stanford, CA, 1963. (Reprinted in *Semantic Information Processing*, M. Minsky (ed.), MIT Press, Cambridge, MA, 1968, pp. 410-417.)
- [Michalski80] R. S. Michalski, "Pattern Recognition as Rule-Guided Inductive Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2, 4 (1980), pp. 349-361.
- [Michalski83] R. S. Michalski, "A Theory and Methodology of Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 83-134.
- [Minsky75] M. L. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, NY, 1975, pp. 211-277.
- [Minton84] S. N. Minton, "Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 251-254.
- [Minton85] S. N. Minton, "Selectively Generalizing Plans for Problem-Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 596-599.
- [Minton86] S. N. Minton, "Overview of the PRODIGY Learning Apprentice," in *Machine Learning: A Guide To Current Research*, T. M. Mitchell, J. G. Carbonell and R. S. Michalski (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 199-202.

- [Minton87] S. N. Minton, J. G. Carbonell, O. Etzioni, C. A. Knoblock and D. R. Kuokka, "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 122-133.
- [Mitchell78] T. M. Mitchell, "Version Spaces: An Approach to Concept Learning," Ph.D. Thesis, Department of Computer Science, Stanford University, Palo Alto, CA, 1978. (Also appears as Technical Report STAN-CS-78-711, Stanford University)
- [Mitchell82] T. M. Mitchell, "Generalization as Search," *Artificial Intelligence* 18, 2 (1982), pp. 203-226.
- [Mitchell83a] T. M. Mitchell, "Learning and Problem Solving," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 1139-1151.
- [Mitchell83b] T. M. Mitchell, P. E. Utgoff and R. Banerji, "Learning by Experimentation: Acquiring and Refining Problem-solving Heuristics," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 163-190.
- [Mitchell85] T. M. Mitchell, S. Mahadevan and L. I. Steinberg, "LEAP: A Learning Apprentice for VLSI Design," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 573-580.
- [Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning* 1, 1 (1986), pp. 47-80.
- [Mooney85] R. J. Mooney and G. F. DeJong, "Learning Schemata for Natural Language Processing," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 681-687.
- [Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555.
- [Mooney88] R. J. Mooney, "A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1988. (Also appears as UILU-ENG-87-2269, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)
- [Mostow83] D. J. Mostow, "Machine Transformation of Advice into a Heuristic Search Procedure," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 367-404.
- [Mostow87] J. Mostow and N. Bhatnager, "Failsafe - A Floor Planner that Uses EBG to Learn from its Failures," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 249-255.

- [Natarajan87] B. Natarajan, "Learning Functions from Examples," Technical Report CMU-TR-87-19, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1987.
- [Neves85] D. M. Neves, "Learning Procedures from Examples and by Doing," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 624-630.
- [Nilsson80] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA, 1980.
- [Novak76] G. S. Novak, "Computer Understanding of Physics Problems Stated in Natural Language," Technical Report NL-30, Ph.D. Thesis, Department of Computer Science, University of Texas at Austin, March 1976.
- [O'Rorke84] P. V. O'Rorke, "Generalization for Explanation-based Schema Acquisition," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 260-263.
- [O'Rorke87a] P. V. O'Rorke, "Explanation-Based Learning via Constraint Posting and Propagation," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1987. (Also appears as UILU-ENG-87-2239, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)
- [O'Rorke87b] P. V. O'Rorke, "LT Revisited: Experimental Results of Applying Explanation-Based Learning to the Logic of Principia Mathematica," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 148-159.
- [Palmer83] M. S. Palmer, "Inference-Driven Semantic Analysis," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 310-313.
- [Paterson78] M. S. Paterson and M. N. Wegman, "Linear Unification," *Journal of Computer and System Sciences* 16, (1978), pp. 158-167.
- [Pazzani85] M. J. Pazzani, "Explanation and Generalization Based Memory," *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, August 1985, pp. 323-328.
- [Pazzani87] M. J. Pazzani, "Inducing Causal and Social Theories: A Prerequisite for Explanation-based Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 230-241.
- [Porter85] B. Porter and D. Kibler, "A Comparison of Analytic and Experimental Goal Regression for Machine Learning," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 555-559.
- [Porter86] B. W. Porter and D. F. Kibler, "Experimental Goal Regression: A Method for Learning Problem-Solving Heuristics," *Machine Learning* 1, 3 (1986), pp. 249-285.
- [Posner68] M. J. Posner and S. W. Keele, "On the Genesis of Abstract Ideas," *Journal of Experimental Psychology* 77, 3 (July 1968), pp. 353-363.

- [Press86] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 1986.
- [Prieditis86] A. E. Prieditis, "Discovery of Algorithms from Weak Methods," *Proceedings of the International Meeting on Advances in Learning*, Les Arcs, Switzerland, 1986, pp. 37-52. (Also to appear in *Machine Learning: An Artificial Intelligence Approach*, Volume III, R. S. Michalski and Y. Kodratoff (eds.), Morgan Kaufman, Los Altos, CA, 1988.)
- [Prieditis87] A. E. Prieditis and J. Mostow, "PROLEARN: Towards a Prolog Interpreter that Learns," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 494-498.
- [Quillian68] M. R. Quillian, "Semantic Memory," in *Semantic Information Processing*, M. L. Minsky (ed.), MIT Press, Cambridge, MA, 1968.
- [Quinlan79] J. R. Quinlan, "Discovering Rules from Large Collections of Examples: A Case Study," in *Expert Systems in the Micro Electronic Age*, D. Michie (ed.), Edinburgh University Press, Edinburgh, Scotland, 1979.
- [Rajamoney85] S. Rajamoney, G. F. DeJong and B. Faltings, "Towards a Model of Conceptual Knowledge Acquisition through Directed Experimentation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985.
- [Rajamoney87] S. Rajamoney and G. F. DeJong, "The Classification, Detection and Handling of Imperfect Theory Problems," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 205-207.
- [Reiter78] R. Reiter, "On Closed World Data Bases," in *Logic and Data Bases*, H. Gallaire and J. Minker (ed.), Plenum Press, NY, 1978.
- [Rendell85] L. Rendell, "Substantial Constructive Induction using Layered Information Compression: Tractable Feature Formation in Search," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 650-658.
- [Rosenbloom86] P. Rosenbloom and J. Laird, "Mapping Explanation-Based Generalization into Soar," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 561-567.
- [Ross84] B. H. Ross, "Reminders and their Effects in Learning a Cognitive Skill," *Cognitive Psychology* 16, (1984), pp. 371-416.
- [Ross88] B. H. Ross, "Reminders in Learning and Instruction," in *Similarity and Analogical Reasoning*, S. Vosniadou and A. Ortony (ed.), Cambridge University Press, Cambridge, England, forthcoming.
- [Rumelhart86] D. E. Rumelhart, G. E. Hinton and J. L. McClelland, "A General Framework for Parallel Distributed Processing," in *Parallel Distributed Processing: Explorations in the Micro-Structure of Cognition*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 46-73.

- [Schank77] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1977.
- [Schank82] R. C. Schank, *Dynamic Memory*, Cambridge University Press, Cambridge, England, 1982.
- [Schank86] R. C. Schank, G. C. Collins and L. E. Hunter, "Transcending Inductive Category Formation in Learning," *Behavioral and Brain Sciences* 9, (1986), pp. 639-686.
- [Schoenfeld82] A. H. Schoenfeld and D. Herrmann, "Problem Perception and Knowledge Structure in Expert and Novice Mathematical Problem Solvers," *Journal of Experimental Psychology: Learning, Memory, and Cognition* 8, 5 (1982), pp. 484-494.
- [Segre85] A. M. Segre and G. F. DeJong, "Explanation Based Manipulator Learning: Acquisition of Planning Ability Through Observation," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, March 1985, pp. 555-560.
- [Segre87a] A. M. Segre, "Explanation-Based Learning of Generalized Robot Assembly Tasks," Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, January 1987. (Also appears as UILU-ENG-87-2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)
- [Segre87b] A. M. Segre, "On the Operationality/Generality Trade-off in Explanation-Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 242-248.
- [Shavlik85a] J. W. Shavlik, "Learning about Momentum Conservation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 667-669.
- [Shavlik85b] J. W. Shavlik and G. F. DeJong, "Building a Computer Model of Learning Classical Mechanics," *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, August 1985, pp. 351-355.
- [Shavlik86a] J. W. Shavlik and G. F. DeJong, "Computer Understanding and Generalization of Symbolic Mathematical Calculations: A Case Study in Physics Problem Solving," *Proceedings of the 1986 Symposium on Symbolic and Algebraic Computation*, Waterloo, Ontario, Canada, July 1986, pp. 148-153.
- [Shavlik86b] J. W. Shavlik and G. F. DeJong, "A Model of Attention Focussing During Problem Solving," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, August 1986, pp. 817-822.
- [Shavlik87a] J. W. Shavlik and G. F. DeJong, "Modeling the Use of Examples to Teach General Concepts in Mathematics-Based Domains," *Third International Conference of Artificial Intelligence and Education*, Pittsburgh, PA, May 1987.

NO-A191 327

GENERALIZING THE STRUCTURE OF EXPLANATIONS IN
EXPLANATION-BASED LEARNING(U) ILLINOIS UNIV AT URBANA
COORDINATED SCIENCE LAB J M SHAYLIK DEC 87

4/4

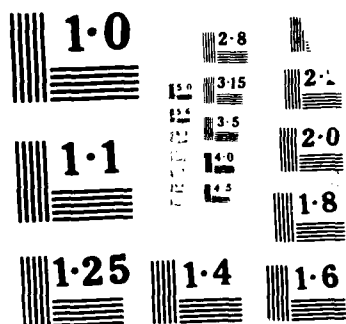
UNCLASSIFIED

UIIU-ENG-07-2276 N00014-86-K-0309

F/G 23/2

NL





- [Shavlik87b] J. W. Shavlik and G. F. DeJong, "Analyzing Variable Cancellations to Generalize Symbolic Mathematical Calculations," *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*, Orlando, FL, February 1987.
- [Shavlik87c] J. W. Shavlik and G. F. DeJong, "An Explanation-Based Approach to Generalizing Number," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 236-238.
- [Shavlik87d] J. W. Shavlik and G. F. DeJong, "BAGGER: An EBL System that Extends and Generalizes Explanations," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 516-520.
- [Shavlik87e] J. W. Shavlik, G. F. DeJong and B. H. Ross, "Acquiring Special Case Schemata in Explanation-Based Learning," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, July 1987, pp. 851-860.
- [Siklossy75] L. Siklossy and D. A. Sykes, "Automatic Program Synthesis from Example Problems," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, U.S.S.R., August 1975, pp. 268-273.
- [Silver83] B. Silver, "Learning Equation Solving Methods from Examples," *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., August 1983, pp. 429-431.
- [Silver86] B. Silver, *Meta-Level Inference*, North-Holland, New York, NY, 1986.
- [Simon83] H. A. Simon, "Why Should Machines Learn?," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Tioga Publishing Co., Palo Alto, CA., 1983, pp. 25-37.
- [Sims87] M. H. Sims, "Empirical and Analytic Discovery in IL," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 274-280.
- [Sleeman82] D. H. Sleeman and J. S. Brown (ed.), *Intelligent Tutoring Systems*, Academic Press, New York, NY, 1982.
- [Smith85] D. E. Smith and M. R. Genesereth, "Ordering Conjunctive Queries," *Artificial Intelligence* 26, 2 (1985), pp. 171-215.
- [Soloway78] E. Soloway, "Learning = Interpretation + Generalization: A Case Study in Knowledge-Directed Learning," Ph. D. Thesis, University of Massachusetts, Amherst, MA, 1978. (Also appears as COINS Technical Report 78-13.)
- [Steier87] D. Steier, "CYPRESS-Soar: A Case Study in Search and Learning in Algorithm Design," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 327-330.
- [Stepp84] R. E. Stepp, "Conjunctive Conceptual Clustering: A Methodology and Experimentation," Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1984.

- [Summers77] P. D. Summers, "A Methodology for LISP Program Construction from Examples," *Journal of the Association for Computing Machinery* 24, (1977), pp. 161-175.
- [Sussman73] G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Report 297, MIT AI Lab, Cambridge, MA, 1973.
- [Sweller85] J. Sweller and G. A. Cooper, "The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra," *Cognition and Instruction* 2, 1 (1985), pp. 59-89.
- [Tadepalli86] P. V. Tadepalli, "Learning in Intractable Domains," in *Machine Learning: A Guide To Current Research*, T. M. Mitchell, J. G. Carbonell and R. S. Michalski (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 337-342.
- [Treitel86] R. Treitel, "Re-arranging Rules," *Proceedings of the Workshop on Knowledge Compilation*, Otter Crest, OR, September 1986.
- [Utgoff84] P. E. Utgoff, "Shift of Bias for Inductive Concept Learning," Ph. D. Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1984.
- [Valiant84] L. G. Valiant, "A Theory of the Learnable," *Communications of the Association for Computing Machinery* 27, 11 (1984), pp. 1134-1142.
- [Vere78] S. A. Vere, "Inductive Learning of Relational Productions," in *Pattern Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (ed.), Academic Press, New York, 1978.
- [Waldinger77] R. Waldinger, "Achieving Several Goals Simultaneously," in *Machine Intelligence* 8, E. Elcock and D. Michie (ed.), Ellis Horwood Limited, London, 1977.
- [Watanabe87] L. Watanabe and R. Elio, "Guiding Constructive Induction for Incremental Learning from Examples," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 293-296.
- [Weld86] D. S. Weld, "The Use of Aggregation in Casual Simulation," *Artificial Intelligence* 30, 1 (1986), pp. 1-34.
- [Wenger87] E. Wenger, *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Morgan-Kaufmann, Los Altos, CA, 1987.
- [Whitehall87] B. L. Whitehall, "Substructure Discovery in Executed Action Sequences," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, September 1987. (Also appears as Technical Report UILU-ENG-87-2256, AI Research Group, Coordinated Science Laboratory.)
- [Wilkins86] D. C. Wilkins, W. J. Clancey and B. G. Buchanan, "ODYSSEUS: A Learning Apprentice," in *Machine Learning: A Guide To Current Research*, T. M. Mitchell, J. G. Carbonell and R. S. Michalski (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 369-374.

- [Winston75] P. H. Winston, "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, NY, 1975, pp. 157-210.
- [Winston82] P. H. Winston, "Learning New Principles From Precedents and Exercises," *Artificial Intelligence* 19, (1982), pp. 321-350.
- [Winston83] P. H. Winston, T. O. Binford, B. Katz and M. Lowry, "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., August 1983, pp. 433-439.
- [Wolff82] J. G. Wolff, "Language Acquisition, Data Compression and Generalization," *Language and Communication* 2, 1 (1982), pp. 57-89.
- [Wolfram83] S. Wolfram, "SMP Reference Manual," Computer Mathematics Group, Inference Corporation, Los Angeles, CA, 1983.

Vita

Jude William Shavlik was born on November 29, 1957 in Manitowoc, Wisconsin. In 1975 he enrolled at the Massachusetts Institute of Technology, where he received a B.S. degree in Electrical Engineering and Computer Science and a B.S. degree in Biology in 1979. He then was awarded a M.S. degree in Molecular Biophysics and Biochemistry from Yale University in 1980. Following that he was employed for several years by the Mitre Corporation, performing simulation analyses of computer networks for NASA. In the fall of 1982 he returned to graduate school, enrolling in the University of Illinois' physics department. The following year he transferred to the computer science department. In the spring of 1984 he became a research assistant for Professor Gerald DeJong in the Coordinated Science Laboratory and began investigating explanation-based learning. He received the Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in January 1988. Currently, he is an assistant professor in the Computer Sciences Department at the University of Wisconsin - Madison.

END

DATE

FILMED

5-88
DTIC